



零起点 Python足彩大数据 与机器学习实盘分析

何海群 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书使用 Python 编程语言、Pandas 数据分析模块、机器学习和人工智能算法对足彩大数据进行实盘分析,设计并发布了开源大数据项目 tfbDat 足彩数据包,汇总了 2010—2017 年全球近 7 万场足球比赛的赛事和赔率数据。此外,还介绍使用 Python 语言抓取网页数据、下载更新 tfbDat 足彩数据包、预测和分析比赛球队的取胜概率,同时提出了检测人工智能算法优劣的“足彩图灵”法则。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有,侵权必究。

图书在版编目(CIP)数据

零起点 Python 足彩大数据与机器学习实盘分析 / 何海群著. —北京:电子工业出版社, 2017.5
ISBN 978-7-121-31074-4

I. ①零… II. ①何… III. ①软件工具—程序设计—应用—足球运动—彩票 IV. ①F719.52-39

中国版本图书馆 CIP 数据核字(2017)第 050287 号

策划编辑:黄爱萍

责任编辑:徐津平

印 刷:

装 订:

出版发行:电子工业出版社

北京市海淀区万寿路 173 信箱 邮编:100036

开 本:787×980 1/16 印张:27.5 字数:523 千字

版 次:2017 年 5 月第 1 版

印 次:2017 年 5 月第 1 次印刷

定 价:99.00 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888, 88258888。

质量投诉请发邮件至 zlts@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式:(010) 51260888-819, faq@phei.com.cn。

前言

从足彩到量化，再从量化到足彩。

生命总是在轮回中，不断完成自我突破与成长壮大。

本书是“Python 量化三部曲”的补充部分。

“Python 量化三部曲”

“Python 量化三部曲”包括：

- 《零起点 Python 大数据与量化交易》（入门课程）；
- 《零起点 Python 量化与机器学习实盘分析》（重点分析 Sklearn）；
- 《零起点 Python 量化与 TensorFlow 深度学习实盘分析》（重点分析 TensorFlow）。

此外还有两部补充作品：

- 《零起点 Python 足彩大数据与机器学习实盘分析》；
- 《零起点 Python 机器学习快速入门》。

较好的 Python 机器学习入门教程

本书中的机器学习算法章节，是目前较系统的 Python 机器学习入门教程，其特点如下。

- 独创的黑箱教学模式，全书无任何抽象理论和深奥的数学公式。
- 首次系统化融合 Sklearn 人工智能软件和 Pandas 数据分析软件，无需直接使用复杂的 Numpy 数学矩阵模块。
- 三位一体的课件模式：图书+开发平台+成套的教学案例。系统讲解，逐步深入。
- 业内第一个系统化的 Sklearn 函数和 API 中文文档，可作为案头工具书随时查阅。
- 基于 Sklearn+Pandas 模式，无需任何理论基础，全程采用 MBA 案例模式，懂 Excel 就可看懂。

这些内容采用独创的黑箱模式和 MBA 案例教学机制，结合一线实战案例，介绍 Sklearn 人工智能模块库和常用的机器学习算法。

进一步学习

本书的读者如果有兴趣可以进一步学习“Python 量化三部曲”的内容，虽然“Python 量化三部曲”的内容是以金融量化分析为主，但基本原理都是相通的，本质上都是数据分析，只是数据源不同，一个是金融数据，另一个是足彩赔率数据。

对于“Python 量化三部曲”的读者而言，本书也有很大的价值，特别是对于入门的读者。

本书有多章关于网络爬虫的内容和具体案例，讲解了提取网络数据的方法，为自己编写自动交易程序的读者提供了一种基于 Web 的操作接口。

网络资源

本书的读者 QQ 群是：264880547（Top 极宽足彩大数据）。

本书有关的程序和数据下载，请浏览网站：TopQuant.vip 极宽量化社区。网站【下载中心】有最新的程序和数据下载地址。

本书在 TopQuant.vip 极宽量化社区设有专栏，对本书和足彩有任何建议的读者，请在社区相关专栏发布信息，笔者会在第一时间进行反馈和答复。

“零起点 Python” 系列丛书

本书继续保持了“零起点 Python”系列丛书的一贯风格，简单实用，书中配有大量图表说明，没有一条数学公式，普通读者只要懂 Word 和 Excel，就能够轻松阅读全书。

- IT 零起点，不需要任何电脑编程基础，会打字、会 Excel 就能看懂本书，利用本书配套的 Python 软件包，轻松学会利用 Python 对股票、足彩数据进行专业分析、量化投资分析。
- 投资零起点，无须购买任何专业软件，本书配套的 zwPython 软件包采用开源模式，提供 100%全功能、全免费的工业级数据分析平台。
- 配置零起点，所有软件、数据全部采用苹果“开箱即用”模式，绿色版本，无须安装，解压即可，直接运行系统。
- 理财零起点，不需要任何专业金融背景，使用通俗易懂的语言，配合大量专业图表和实盘操作案例，轻松掌握各种量化投资策略。
- 数学零起点，全书没有任何复杂的数学公式，只有最基本的加、减、乘、除，易于理解。

关于足彩的几个误区

近年来，大数据产业、人工智能行业风生水起，可国内足彩大数据的专业研究还处于冷门和偏门，这其中有很多关于足彩领域的误区。

很多主流学者，到现在都看不起足彩，认为是赌博，这个我们不讨论。最近国家级别的彩票大数据研究中心已经正式成立了，资本的力量是无穷的。

为此，笔者做了一个简单的小结，以正视听。

- 足彩虽然容易与赌球混淆，但却是最好的大数据研究对象，没有之一。
- 微软、百度和谷歌等公司目前都有专业团队在做足彩大数据研究，并定期发布结果。
- 足彩相当于十倍配资的股票。
- 国内足彩赔率的确很低，差不多是全球最低的，比欧洲平均低 10%左右。

- 彩票和股票的发明人据说都是同一个英国爵士。
- 必胜足彩交易所成立当年获得了英国 MBA 商业创新大奖。
- 高盛公司多年前就开始进行足彩套利业务，维基百科中有介绍。
- 极宽黑天鹅（红牛吧）足彩是业内首家公开进行实盘测试的足彩大数据模型。
- 黑天鹅算法在业内率先以“盈利率”而不是“胜率”测试足彩算法模型。

致谢

虽然很多网友在笔者博客中留言，建议笔者早日完成这本书的写作，但本书的创作和正式出版还是经历了许多波折。

如今本书终于出版，在此，要特别感谢电子工业出版社的黄爱萍和戴新编辑，感谢她们在选题策划和稿件整理方面所做的大量工作。

同时，在本书创作过程中，极宽开源量化团队和培训班的全体成员提出很多宝贵的意见，并对部分课件程序做了中文注解。

特别是吴娜、余勤、王硕三位同学，为极宽开源量化文库和 zwQuant 开源量化软件编写文档，并在团队成员管理方面做了大量工作，对他们的付出表示感谢。

何海群（字王）

北京极宽科技有限公司 CTO

2017 年 3 月 25 日

目 录

第 1 章 足彩与数据分析	1
1.1 “阿尔法狗”与足彩	1
1.2 案例 1-1：可怕的英国足球	3
1.3 关于足彩的几个误区	7
1.4 足彩·大事件	8
1.5 大数据图灵（足彩）原则	10
1.6 主要在线彩票资源	11
1.7 主要在线足彩数据源	15
1.8 足彩基础知识	17
1.9 学习路线图	18
第 2 章 开发环境	19
2.1 数据分析首选 Python	19
2.1.1 大数据，why Python	19
2.1.2 入门简单，功能强大	21
2.1.3 难度降低 90%，性能提高 10 倍	23
2.1.4 “零对象”编程模式	24
2.2 用户运行平台	25
2.3 程序目录结构	26
2.4 tfbDat 足彩数据包	27

2.5	Spyder 编辑器界面设置	28
2.5.1	开发环境界面设置	28
2.5.2	代码配色技巧	29
2.5.3	图像显示配置	31
2.5.4	重剑无锋	32
2.6	Notebook 模式	34
2.7	模块库控制面板	36
2.7.1	模块库资源	37
2.7.2	模块库维护更新	37
2.7.3	系统关联	38
2.8	使用 pip 命令更新模块库	39
2.8.1	pip 常用命令	39
2.8.2	进入 Python 命令行模式	41
2.8.3	pip 安装模板	41
2.8.4	pip 参数解释	42
2.8.5	pip-install 参数选项	43
第 3 章	入门案例套餐	45
3.1	案例 3-1: 第一次编程, “hello,ziwang”	45
3.1.1	简单调试	46
3.1.2	控制台复位	47
3.2	案例 3-2: 增强版 “hello,ziwang”	47
3.3	案例 3-3: 列举系统模块库清单	49
3.4	案例 3-4: 常用绘图风格	50
3.5	案例 3-5: Pandas 常用绘图风格	52
3.6	案例 3-6: 常用颜色表 cors	53
第 4 章	足彩量化分析系统	55
4.1	功能简介	55
4.1.1	目录结构	56
4.1.2	TFB 安装与更新	56

4.2	TFB 主体框架	57
4.2.1	模块构成	57
4.2.2	Top-Base 极宽基础模块库	57
4.2.3	Top-Football 极宽足彩专业模块库	58
4.2.4	tfbDat 极宽足彩数据包	59
4.2.5	量化系统模块构成	60
4.2.6	案例 4-1: 赔率文件切割	61
4.2.7	案例 4-2: 批量切割数据文件	64
4.3	tfbDat 数据结构	66
4.3.1	案例 4-3: tfb 数据格式	67
4.3.2	gid 基本比赛数据格式	67
4.3.3	xdatt 赔率数据格式	69
4.4	足彩基本数据分析	73
4.4.1	案例 4-4: 比赛数据基本图表分析	73
4.4.2	案例 4-5: 比赛数据进阶图表分析	77
4.4.3	案例 4-6: 比赛数据年度图表分析	80
4.4.4	案例 4-7: 比赛数据时间细分图表分析	81
4.5	胜、平、负数据分析	88
4.5.1	案例 4-8: 胜、平、负数据分析	88
4.5.2	@修饰符	88
4.5.3	胜、平、负分析	90
4.6	赔率数据分析	91
4.6.1	案例 4-9: 赔率分析	91
4.6.2	扩充 dr_gid_top10 绘图函数	92
4.6.3	赔率对比	93
第 5 章	常用数据分析工具	96
5.1	Pandas 数据分析软件	96
5.1.1	Pandas 简介	96
5.1.2	案例 5-1: Pandas 常用统计功能	99
5.2	科学计算	104

5.3	人工智能	105
5.4	NLTK 语义分析	107
5.5	数据清洗统计分析	109
5.6	数据可视化	109
第 6 章	辅助工具	114
6.1	性能优化	114
6.1.1	Numexpr 矢量加速库	115
6.1.2	Numba 支持 GPU 的加速模块库	115
6.1.3	Blaze 大数据优化模块库	115
6.1.4	Pyston 加速模块	116
6.1.5	PyPy 加速模块	116
6.1.6	Cython	116
6.1.7	其他优化技巧	117
6.2	网页信息抓取	117
6.2.1	Requests 人性化的网络模块	118
6.2.2	Scrapy 网页爬虫框架	118
6.2.3	Beautiful Soup 4	119
6.3	其他工具模块	120
6.3.1	Logging 日志模块	120
6.3.2	Debug 调试工具	121
6.3.3	re 正则表达式	121
6.3.4	并行编程	122
6.4	网络辅助资源	123
6.5	arrow 优雅简捷的时间模块库	125
6.5.1	案例 6-1: arrow 入门案例	126
6.5.2	创建 arrow 时间对象	128
6.5.3	创建时间戳	128
6.5.4	arrow 属性	129
6.5.5	replace 替换和 shift 位移	130
6.5.6	format 格式化参数	130

6.5.7	时间转换	131
6.5.8	短命令	131
6.5.9	人性化	131
6.5.10	范围和跨度	132
6.5.11	工厂模式	133
6.5.12	Token 特殊字符	133
第 7 章	网络足彩数据抓取	135
7.1	500 彩票网站数据接口的优势	135
7.1.1	案例 7-1: 抓取赔率数据网页	136
7.1.2	网页数据实战操作技巧	139
7.2	网页解析的心灵鸡汤	141
7.2.1	BS4 四大要素三缺一	142
7.2.2	Tag 标签对象	142
7.2.3	案例 7-2: Tag 标签对象	142
7.2.4	案例 7-3: Tag 标签对象数据类型	145
7.2.5	NavigableString 导航字符串	149
7.2.6	BeautifulSoup 复合对象	149
7.2.7	Comment 注释对象	150
7.2.8	案例 7-4: BS4 查找匹配功能	150
7.2.9	BS4 节点遍历功能	154
7.3	足彩基本数据抓取	155
7.3.1	案例 7-5: 分析网页比赛数据	155
7.3.2	案例 7-6: 提取网页比赛数据	157
7.3.3	gid 比赛基本数据结构	159
7.3.4	案例 7-7: 提取比赛得分	161
7.3.5	案例 7-8: 提取球队 id 编码	164
7.3.6	案例 7-9: 抓取历年比赛数据	167
7.3.7	案例 7-10: 流程图工具与 Python	171
7.3.8	实盘技巧	172
7.3.9	案例 7-11: 进程池并发运行	174

7.4	批量抓取足彩网页数据实盘教程	177
7.4.1	案例 7-12: 批量抓取赔率数据	177
7.4.2	fb_gid_getExt 扩展网页下载函数	178
7.4.3	bars 节点数据包与 pools 彩票池	178
7.4.4	抓取扩展网页	180
7.5	足彩赔率数据抓取	181
7.5.1	gid 与赔率数据网页	181
7.5.2	案例 7-13: 提取赔率数据	184
7.5.3	赔率数据与结构化数据	186
7.5.4	瀑布流数据网页与小数据理论	189
第 8 章	足彩数据回溯测试	191
8.1	TFB 系统构成	192
8.1.1	TFB 系统模块结构	192
8.1.2	Top-Base 极宽基础模块库	192
8.1.3	Top-Football 极宽足彩专业模块库	193
8.2	实盘数据更新	194
8.2.1	案例 8-1: 实盘数据更新	194
8.2.2	实盘要点: 冗余	195
8.2.3	实盘要点: 耐心	196
8.2.4	实盘要点: 数据文件	197
8.2.5	main_get 函数	197
8.3	变量初始化	199
8.3.1	全局变量与类定义	201
8.3.2	彩票池内存数据库	202
8.3.3	案例 8-2: 内存数据库&数据包	204
8.4	回溯测试	205
8.4.1	案例 8-3: 回溯	206
8.4.2	main_bt 回溯主入口	207
8.4.3	案例 8-4: 实盘回溯	209
8.4.4	彩票池与统计池	211

8.4.5	poolTrd 下单交易数据	212
8.4.6	poolRet 回报记录数据	213
8.4.7	实盘足彩推荐分析	214
8.4.8	实盘回报分析	214
8.4.9	全数据分析与足彩数据集	215
8.5	bt_main 回溯主函数	216
8.5.1	bt_1dayMain 单日回溯函数	218
8.5.2	赔率数据合并函数	219
8.5.3	单日回报分析函数	220
8.5.4	单日回报分析	221
8.5.5	单场比赛回报分析	223
8.6	sta01 策略的大数据分析	224
8.6.1	一号策略函数	226
8.6.2	超过 100%的盈利策略与秘诀	227
8.6.3	统计分析	228
8.6.4	回溯时间测试	229
8.6.5	bt_main_ret 总回报分析	230
第 9 章	参数智能寻优	232
9.1	一元参数寻优	233
9.1.1	案例 9-1: 一号策略参数寻优	233
9.1.2	一元测试函数	234
9.1.3	测试结果数据格式	236
9.1.4	案例 9-2: 一元参数图表分析	237
9.2	策略函数扩展	241
9.2.1	扩展一号策略函数	241
9.2.2	案例 9-3: 一号扩展策略	242
9.2.3	案例 9-4: sta10 策略	244
9.3	二元参数寻优	246
9.3.1	案例 9-5: sta10 参数寻优	246
9.3.2	案例 9-6: 二元参数图表分析	248

9.4	策略 310 准多因子策略	252
9.4.1	案例 9-7: 数据预处理	254
9.4.2	案例 9-8: 策略 310 参数寻优	257
9.4.3	案例 9-9: 策略 310 图表分析	259
9.4.4	案例 9-10: 策略 310	264
第 10 章	Python 人工智能入门与实践	266
10.1	从忘却开始	266
10.2	Iris 经典爱丽丝	269
10.2.1	案例 10-1: 经典爱丽丝	270
10.2.2	案例 10-2: 爱丽丝进化与矢量化文本	272
10.3	AI 操作流程	273
10.3.1	机器学习与测试数据集	274
10.3.2	机器学习运行流程	274
10.3.3	经典机器学习算法	275
10.3.4	黑箱大法	275
10.3.5	数据切割函数	276
10.3.6	案例 10-3: 爱丽丝分解	277
10.3.7	案例 10-4: 线性回归算法	281
第 11 章	机器学习经典算法案例 (上)	286
11.1	线性回归	286
11.2	逻辑回归算法	293
11.2.1	案例 11-1: 逻辑回归算法	294
11.3	朴素贝叶斯算法	296
11.3.1	案例 11-2: 贝叶斯算法	297
11.4	KNN 近邻算法	299
11.4.1	案例 11-3: KNN 近邻算法	301
11.5	随机森林算法	302
11.5.1	案例 11-4: 随机森林算法	306
第 12 章	机器学习经典算法案例 (下)	308
12.1	决策树算法	308

12.1.1	案例 12-1: 决策树算法	310
12.2	GBDT 迭代决策树算法	311
12.2.1	案例 12-2: GBDT 迭代决策树算法	312
12.3	SVM 向量机	313
12.3.1	案例 12-3: SVM 向量机算法	315
12.4	SVM-cross 向量机交叉算法	316
12.4.1	案例 12-4: SVM-cross 向量机交叉算法	317
12.5	神经网络算法	318
12.5.1	经典神经网络算法	319
12.5.2	Sklearn 神经网络算法	320
12.5.3	人工智能学习路线图	320
12.5.4	案例 12-5: MLP 神经网络算法	321
12.5.5	案例 12-6: MLP_reg 神经网络回归算法	323
第 13 章	机器学习组合算法	326
13.1	CCPP 数据集	326
13.1.1	案例 13-1: CCPP 数据集	327
13.1.2	案例 13-2: CCPP 数据切割	328
13.1.3	数据切割函数	330
13.1.4	案例 13-3: 读取 CCPP 数据集	331
13.1.5	数据读取函数	333
13.2	机器学习统一接口函数	334
13.2.1	案例 13-4: 机器学习统一接口	334
13.2.2	统一接口函数	336
13.2.3	机器学习算法代码	338
13.2.4	效果评估函数	339
13.2.5	常用评测指标	340
13.3	批量调用机器学习算法	341
13.3.1	案例 13-5: 批量调用	341
13.3.2	批量调用算法模型	344
13.4	一体化调用	345

13.4.1	案例 13-6: 一体化调用	345
13.4.2	一体化调用函数	346
13.5	模型预制与保存	348
13.5.1	案例 13-7: 储存算法模型	348
13.5.2	模型保存函数	350
13.5.3	模型预测函数	350
13.5.4	案例 13-8: 批量储存算法模型	351
13.5.5	批量模型储存函数	353
13.5.6	案例 13-9: 批量加载算法模型	353
13.6	机器学习组合算法	357
13.6.1	案例 13-10: 机器学习组合算法	357
13.6.2	机器学习组合算法函数	359
第 14 章	足彩机器学习模型构建	361
14.1	数据整理	361
14.1.1	案例 14-1: 赔率数据合成	362
14.1.2	案例 14-2: 按年切割赔率数据	365
14.1.3	案例 14-3: 累计切割赔率数据	365
14.2	年度足彩赔率模型	366
14.2.1	案例 14-4: 2016 年度足彩赔率模型组	367
14.2.2	案例 14-5: 年度多字段足彩赔率模型组	370
14.3	累计足彩赔率模型	373
14.3.1	案例 14-6: 累计 2016 足彩赔率模型组	373
14.3.2	案例 14-7: 累计多字段足彩赔率模型组	376
14.3.3	足彩算法模型文件	379
第 15 章	足彩机器学习模型验证	381
15.1	年度赔率模型验证	381
15.1.1	案例 15-1: 年度赔率模型验证	381
15.1.2	案例 15-2: 多字段年度赔率模型验证	383
15.2	累计赔率模型验证	385

15.2.1	案例 15-3: 累计赔率模型验证	385
15.2.2	案例 15-4: 多字段累计赔率模型验证	386
15.3	年度组合模型验证	388
15.3.1	案例 15-5: 年度组合模型验证	388
15.3.2	案例 15-6: 多字段年度组合模型验证	391
15.3.3	案例 15-7: 全字段年度组合模型验证	391
15.3.4	年度组合模型测试数据对比分析	392
15.4	累计组合模型验证	393
15.4.1	案例 15-8: 年度组合模型验证	393
15.4.2	案例 15-9: 多字段年度组合模型验证	394
15.4.3	累计组合模型测试数据对比分析	394
第 16 章	结果数据分析	397
16.1	神秘的 df9	397
16.1.1	案例 16-1: 调试模式	397
16.1.2	神秘的 df9 结果数据变量	400
16.2	盈利率分析	402
16.2.1	案例 16-2: 盈利率计算	402
第 17 章	机器学习足彩实盘分析	407
17.1	回溯主入口	408
17.1.1	案例 17-1: 策略 sta01	409
17.1.2	结果文件解读	409
17.1.3	数据字段分析	411
17.2	机器学习与回溯分析	412
17.2.1	案例 17-2: Log 回归策略足彩分析	414
17.2.2	Log 回归策略函数	415
17.2.3	案例 17-3: 30 天 Log 回归策略足彩分析	418
17.2.4	数据文件分析	420
17.2.5	足彩推荐	421

1

第 1 章

足彩与数据分析

1.1 “阿尔法狗”与足彩

百度百科中“阿尔法狗”词条的解释如下：

“阿尔法狗”一般指阿尔法围棋。阿尔法围棋（AlphaGo）是一款围棋人工智能程序，由谷歌（Google）旗下 DeepMind 公司的戴密斯·哈萨比斯、大卫·席尔瓦、黄士杰与他们的团队开发，其主要工作原理是“深度学习”。

2016 年 3 月，该程序与围棋世界冠军、职业九段选手李世石进行人机大战，并以 4:1 的总比分获胜。2016 年年末和 2017 年年初，该程序在中国棋类网站上以“大师”（Master）为注册账号与中、日、韩数十位围棋高手进行快棋对决，连续 60 局无一败绩。不少职业围棋手认为，阿尔法围棋的棋力已经达到甚至超过围棋职业九段水平，在世界职业围棋排名中，其等级分曾经超过人类排名第一的棋手柯洁。

2017 年 1 月，谷歌 DeepMind 公司 CEO 哈萨比斯在德国慕尼黑 DLD（数字、生活、设计）创新大会上宣布推出真正 2.0 版本的阿尔法围棋（AlphaGo）。其特点是摒弃了人类棋谱，只靠深度学习的方式成长起来挑战围棋的极限。

现代社会已经进入后互联网时代，信息资源随手可得，大数据产业风生水起，科技创新层出不穷，不过类似工业革命、卫星登月、原子弹爆炸、Internet 信息高速公路等级别的重大科技突破却一直没有出现，甚至有学者认为，目前的社会处于科

技停滞阶段。

直到 2016 年 3 月，谷歌公司的“阿尔法狗”横空出世，与围棋世界冠军、职业九段选手李世石进行人机大战，并以 4:1 的总比分获胜。

这个结果震惊了整个社会，特别是学术界，人工智能领域的从业人员，即使最乐观的学者，在 IBM 公司的“深蓝”战胜国际象棋冠军以后，过去一直都认为：围棋是人类智慧的最后堡垒，19×19 的围棋棋盘矩阵，可以衍生出天文数字的组合变化，至少在五十年内，受计算机技术的限制，人工智能无法达到人类职业选手的标准。

没想到，“阿尔法狗”只用几年时间不仅战胜了围棋职业选手，而且战胜了人类的围棋冠军。

以下资料摘自网络：

（1）AlphaGo 程序首席设计师黄士杰。

黄士杰从小热爱围棋，在中国台湾师大读书时创办了学校的围棋社，曾获台湾大专杯围棋赛冠军，拥有业余六段棋力，是“当时大专生最强”；2010 年，他在读博士的最后一年，开发了围棋软件“Erica”（艾丽卡），参加国际电脑奥林匹亚竞赛（International Computer Games Association，简称 ICGA），击败了当时围棋 AI 公认最强程式“Zen”（禅），引起轰动。黄士杰是 DeepMind 公司团队中的重要成员，是两位首席工程师之一。

Aja Huang（黄士杰）的弈城账号为 DeepMind，据网友调查，AlphaGo 曾经用该账号做过测试。

（2）AlphaGo 之父：一个有着一半华人血统的英国天才哈萨比斯。

1976 年 7 月，哈萨比斯出生于英国伦敦，母亲是新加坡华人，父亲来自希腊的塞浦路斯。17 岁时，在一个游戏设计比赛中，哈萨比斯获得第二名，进入著名的牛蛙游戏软件公司实习，参与设计和开发游戏《主题公园》（Theme Park）。

1998 年，哈萨比斯成立了自己的游戏软件公司（Elixir Studios）。这家拥有 60 人的游戏公司，发布了包括《革命》和《魔鬼天才》等很多游戏软件。

1999 年，年仅 23 岁的哈萨比斯第一次参加了“智力奥林匹克运动会”，这是一个专门为天才较量智力的国际比赛。哈萨比斯连续参加了 4 年，赢了 5 次。

2009 年，因其在游戏设计上的成就，哈萨比斯被选为英国皇家艺术协会的成员。

2011 年，哈萨比斯成立 DeepMind Technologies 公司，其目标是“解决智能问题”。

（3）低调的 DeepMind 公司。

DeepMind 公司一直保持低调，直到 2013 年 12 月，DeepMind 公司首次参加业

界领先的机器学习研究大会时，DeepMind 的研究人员演示他们的软件，一开始就令人惊艳。此前从未有人演示过具备这种能力的软件，即可以从零开始学习和掌握如此复杂的任务。一个月后，谷歌公司重金收购了 DeepMind。再后来，就是我们今天看到的哈萨比斯所创造的历史：他领导开发的人工智能 AlphaGo 打败了世界顶尖的围棋选手。

“阿尔法狗”程序虽然神秘，但其核心算法却很简单，源自古老的 Monte Carlo（蒙特卡罗）算法。

2006 年，欧洲数学家 Rémi Coulomb、Kocsis 和 Szepervari 等学者，在研究围棋程序时，结合蒙特卡罗算法与对手树搜索算法，设计出一种全新的算法：MCTS（蒙特卡罗树搜索算法）。

MCTS 全称 Monte Carlo Tree Search，即蒙特卡罗树搜索算法，是一种在人工智能问题中做出最优决策的方法，它结合了随机模拟的一般性和树搜索的准确性。

而古老的蒙特卡罗算法，正是源自世界上最古老的国际大赌场蒙特卡罗。事实上，现代统计学、博弈学甚至金融领域的量化投资，都不同程度源自赌场和蒙特卡罗算法。

1.2 案例1-1：可怕的英国足球

据说，目前欧洲的顶级数学家 90% 都是被欧洲博彩公司雇佣的，而足彩，更是博彩行业当中最重要的组成部分之一，而英国的博彩公司更是雄霸天下。

口说无凭，有图为证，案例 1-1 通过分析 2010—2016 年近 7 万场足球比赛的数据，用数字和图表来说明这一点。

案例 1-1 的文件名是 `zc101_gid01des.py`，核心代码如下：

```
def gid_anz_top10(df, ksgn):  
  
    xn9=len(df['gid'])  
    d10=df[ksgn].value_counts()[:10];print(d10)  
  
    #  
    #---set chinese font  
    mpl.rcParams['font.sans-serif'] = ['SimHei'] #指定默认字体  
    mpl.rcParams['axes.unicode_minus'] = False #解决保存图像是负号 '-' 显示
```

为方块的问题

```
#
d10.plot(kind = 'bar',rot=0,color=zsyz.cors_brg)
plt.show()
#
dsum=d10.sum()
d10['other']=xn9-dsum
k10=np.round(d10/xn9*100,decimals=2)

k10.plot(kind = 'pie',rot=0,table=True)
plt.show()

#-----
rs0='/tfbdat/'
fgid=rs0+'gid2017.dat'
df=pd.read_csv(fgid,index_col=False,dtypes=str,encoding='gb18030')
print(df.tail())
print('\n',df.describe())
#
gid_anz_top10(df,'gset')
```

案例 1-1 的程序读者看不懂没关系，这里只是简单介绍，后面的章节会有具体讲解。

案例 1-1 中的 gid2017 数据包和本书其他的数据包都是基于实盘数据的，在不断实时更新中，具体的案例运行结果在细节上和具体数字上可能会有所不同，这属于正常现象，请读者不要介意。

现在我们先看看案例 1-1 的部分运行结果。

案例 1-1 调用自定义的 gid_anz_top10 分析函数，分析 gid 比赛数据各个数据列排名前十位的数据，也就是常说的 Top10 数据分析。

案例 1-1 运行结果比较长，我们将分段介绍，第一部分是显示 gid 数据文件的尾部数据和调用 Pandas 的 describe 快速统计汇总函数，对 gid 数据文件做简单的统计：

```
gid gset mplay mtid gplay gtid qj qs qr kend kwin kwinrq
tweek      tplay      tsell
68517  632918   智甲   巴勒人  2032  康塞普西  2823  -1  -1  0   0  -1
-1      0  2017-02-05  2017-02-06  00:55:00
```

```

68518 632919 智甲 天主大 1718 基约塔 6127 -1 -1 0 0 -1
-1 0 2017-02-05 2017-02-06 00:55:00
68519 633032 阿超杯 河床 864 拉努斯 963 -1 -1 0 0 -1
-1 6 2017-02-04 2017-02-05 00:55:00
68520 634245 非洲杯 布基纳 73 加纳 60 -1 -1 0 0 -1
-1 6 2017-02-04 2017-02-05 00:55:00
68521 634246 非洲杯 埃及 41 喀麦隆 5 -1 -1 0 0 -1 -1
0 2017-02-05 2017-02-06 00:55:00

gid gset mplay mtid gplay gtid qj qs qr
kend kwin kwinrq tweek tplay tsell
count 68522 68521 68521 68517 68521 68517 68522 68522 68522
68522 68522 68522 68522 68522 68522
unique 68522 154 2007 1724 1958 1681 14 11 1
2 4 1 7 2530 14785
top 438989 英甲 布里斯 653 布里斯 653 1 1 0
1 3 -1 6 2016-05-08 2013-02-08 23:30:00
freq 1 3807 277 218 280 220 22219 23830 68522
68365 30810 68522 22940 103 131

```

新版本的 `describe` 函数对统计分析结果做了简化，默认没有分位数，只有简单的四行：`count` 数据总数、`unique` 无重复总数、`top` 最多数据和 `freq` 最多的数据频率。

由案例 1-1 的输出信息，我们简单分析 `gid` 比赛数据，可以得出以下结论。

- 2010 年 1 月—2017 年 1 月，共有近 7 万场比赛。
- 在 `gset` 联赛数据中，共有 154 种不同的联赛，排名第一的是英甲，共有 3807 场比赛。
- 在球队中，按名称来看，主队 `mplay` 有 2007 支，客队有 1958 支；按球队 `id` 来看，主队 `mtid` 有 1724 支，客队 `gtid` 有 1681 支。这是由于翻译和历史原因，部分国外球队有几个不同的名称，但球队的 `id` 是唯一的。有趣的是，不管是主队还是客队，比赛次数最多的都是英甲的布里斯球队。
- `qj` 进球数据有 14 种不同情况，其中最多的是进 1 球，有 22219 场。
- `qs` 失球数据有 11 种不同情况，其中最多的是失 1 球，有 23830 场。
- `tweek` 数据列显示星期六的比赛最多，有 22940 场。
- 按 `tplay` 比赛日期来看，2016-05-08 的比赛场次最多，当天有 103 场比赛。
- 按 `tsell` 博彩销售截止日期来看，2013-02-08 销售的比赛场次最多，当天有 131 场比赛。

案例 1-1 输出信息的第二部分是 Top10 的联赛信息数据：

英甲	3807
英冠	3779
日职乙	3033
阿甲	2744
巴甲	2642
英超	2598
西甲	2597
法甲	2595
意甲	2548
法乙	2378

英国不愧是足球强国，比赛场次最多的第一名英甲、第二名英冠都是英国的足球比赛，令人意外的是排名第三的是日本的足球联赛日职乙，比阿根廷、巴西的足球比赛场次还多。

由日本的数据可以看出，人口基数和经济实力对于足球联赛的支持还是很大的，中国的经济实力和综合国力目前已经远远超过日本，人口基数更是日本的数倍，从职业联赛的基数方面看，超越日本应该是不难的，有了数量方面的突破，才会有质的蜕变。

案例 1-1 输出信息的第三部分是联赛数据 Top10 的柱形图，如图 1-1 所示。

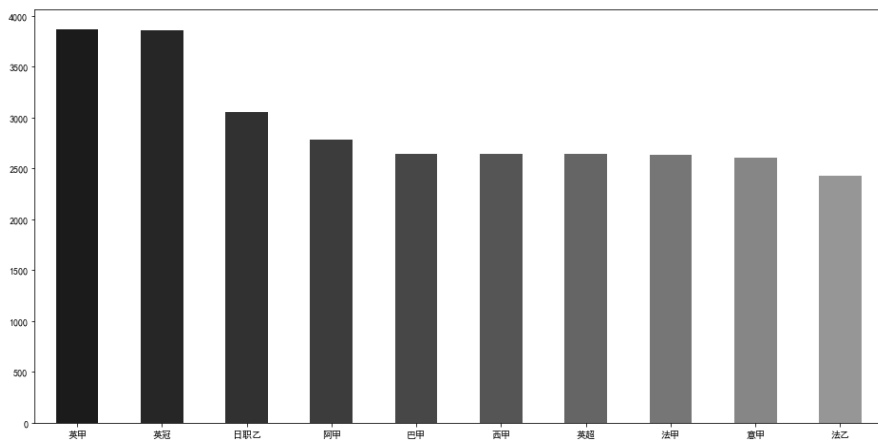


图 1-1 联赛数据 Top10 的柱形图

案例 1-1 输出信息的第四部分是联赛数据 Top10 的比例图，如图 1-2 所示。

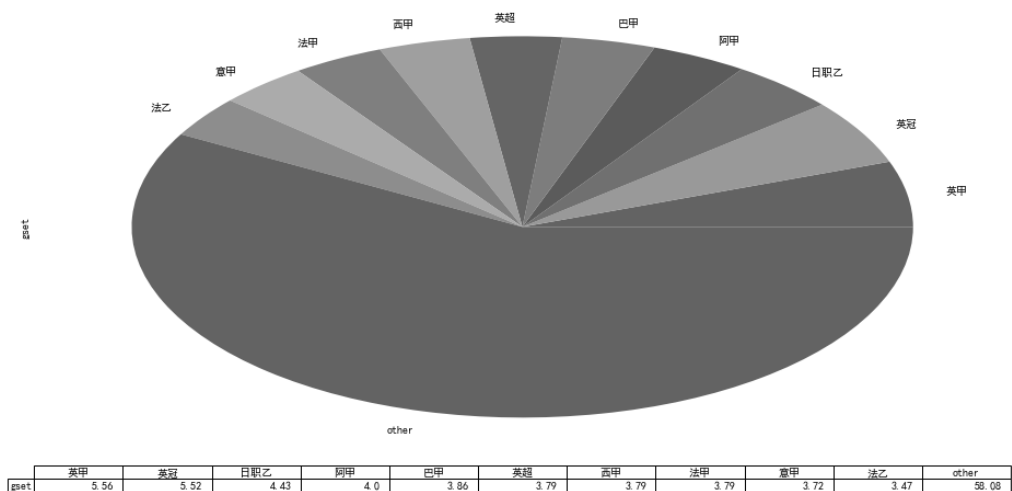


图 1-2 联赛数据 Top10 的比例图

为了进一步说明，第四部分的图形做了以下部分优化。

- 采用计算过的百分数据，表示相关的联赛比例。
- 德甲和英超的比赛场次合计超过了比赛总场次的 11%。
- 排名前 10 的联赛占据了 42% 的比赛场次，其他联赛的比赛场次合计约 58%。

这些数据只是对 gid 比赛基本数据最简单的分析，还谈不上大数据，不过已经可以获得许多有意思的结论。

由以上数据和图表分析可以看出，英国在足彩行业占据 10% 以上的份额，不愧是老牌足球强国。

1.3 关于足彩的几个误区

近年来，大数据产业、人工智能行业风生水起，可我国足彩大数据的专业研究还是冷门和偏门，这其中有很多关于足彩领域的误区。

很多主流学者，到现在都看不起足彩，认为是赌博，这个我们不讨论，最近国家级别的彩票大数据研究中心已经正式成立了，资本的力量是无穷的。

为此，笔者做了一个简单的小结。

- 足彩虽然容易与赌球混淆，但却是最好的大数据研究对象，没有之一。

- 微软、百度、谷歌等公司目前都有专业团队在做足彩大数据研究，并定期发布。
- 通俗地说，足彩相当于十倍配资的股票。
- 我国足彩赔率的确很低，差不多是全球最低的，比欧洲平均低 10%左右。
- 彩票和股票的发明人据说都是同一个英国爵士。
- 必胜足彩交易所成立当年获得了英国 MBA 商业创新大奖。
- 高盛公司多年前就开始进行足彩套利业务，维基百科中有介绍。
- 极宽黑天鹅（红牛吧）足彩是业内首家公开进行实盘测试的足彩大数据模型。
- 黑天鹅算法在业内率先以“盈利率”而不是“胜率”测试足彩算法模型。

1.4 足彩·大事件

下面笔者介绍一下自己的足彩研究历程，本节内容摘自笔者博客，以大事件的方式节选如下。

- 笔者虽然马马虎虎，却做过不少很好的项目，像现在的黑天鹅足彩算法，可以说在行业内算得上 Top10 了。
- 黑天鹅项目最早源于笔者的验证码项目，当时叫 z-SPO: z 粒子算法，2012 年的博客中有提过。
- 在做验证码识别项目时，笔者想将新浪、SOHU、网易、天涯这些网站分类，识别角度可能更高，于是就写了一个自动分类算法。
- 算法其实很简单，就是“二选一”模式：首先，是新浪的验证码，类型标记为“新浪”，不是的标记为“其他”；然后，对“其他”进行二次分类，是 SOHU 的，标记为“搜狐”，不是的标记为“其他”。这样，几个循环下来，自动分类就完成了。
- 令人吃惊的是，这个程序很简单，准确度却相当高，差不多达到 90%，可以完成几百种不同风格验证码的自动分类。
- 因为是“二选一”模式，很容易就联想到股市的涨与跌。这方面，读者可以看笔者的博客《文科生、易经与大数据》，这里就不展开说明了。
- 下载了历年的股票数据，当时是 2010 年，从 2004—2010 年的数据都有，这些数据现在还在网盘里面，不过格式忘记了，要看看源程序。
- 对股市研究了一年，各种公式、指标、算法差不多都测试过，没有一个靠谱的。

- 笔者做项目，喜欢看一些相关的外围书，特别是相关的历史书，在研究股市数据的时候，笔者发现股市和足彩的发明人都是同一个英国贵族。
- 通过维基百科，笔者知道了必发交易所获得过商业创新大奖，知道了高盛足彩套利的故事。
- 在 2012 年，朋友叫笔者去深圳做项目，在高丽，深圳大学城附近，笔者喜欢一个人在街上溜达，笔者发现离高丽地铁站十分钟的路程内，至少有三十家彩票投注站，比药店、银行还要多，仅次于手机店，要知道，深圳的门面租金差不多每月一万元。
- 于是，在网上查询，发现国内正规的彩票网站差不多有上百家，而且大部分都有上市公司背景。
- 查看彩票网站的现金流，基本上都是十亿元、百亿元级别，很少有千万元级别的（2015 年，深圳查封一家足彩网站，一年的流水就是上千亿元）。
- 笔者做过很多互联网公司的方案，知道很多互联网公司看起来风光，可实际上就几千万元、几百万元的现金在周转。
- 再想到高盛、必发的故事，于是开始研究足彩。
- 起步很痛苦，因为笔者从不看足球，现在也是，更不用说“3、1、0、胜、平、负”了，还有什么大球、小球、上盘、下盘。
- 刚开始只是看，不掏钱，很多细节总是搞混，好在买足彩才 2 元。笔者买足彩是看赔率、看数字，看那个数字顺眼，就投哪个。
- 交了不少学费后，开始知道有合买了，而且运气不错，跟单开始时有连续几个 4~5 倍的胜盘，不过一个月左右就全亏了，于是换人合买，换了很多，还是不靠谱。
- 这期间交了不少学费，基本上每个环节都交过，因为是真金白银，虽然不多，但都有印象。所以，一定要做实盘练习，一定要注意细节。
- 没有实盘，量化训练就没有任何意义，有了实盘，细节上吃几次亏，就会注意了，这总比正式投资出问题好。
- 当然，这时候，对于足彩的数据分析也没停。足彩也有好多所谓的必胜公式，试了很多个，不靠谱，再用经典的教科书算法、机器学习、数据挖掘，还是不靠谱。
- 最终，转来转去，还是笔者自己的算法最靠谱。笔者认为，不是笔者的算法靠谱，可能是，一方面靠谱的算法一般机构不会公开，另一方面公开的算法大部分是限制模型，是有很多前提的，部分没有前提的，准确度又不行。

- 2014—2015 年，大数据风起云涌，笔者跟风，把 zwPython 给开源了。
- 2015 年，发布我国首个足彩开源数据包 zc-dat，收录近 5 万场比赛数据。
- 2016 年，发布我国首个开源量化软件 zwQuant 和 zwDat 金融数据包。
- 2016 年，组建 TopQuant.vip 北京极宽科技公司。
- 2016 年 12 月，笔者在电子工业出版社正式出版《零起点 Python 大数据与量化交易》。
- 2017 年，笔者创作本书。
- 2017 年，发布 Top Quant for Football（简称 TFB，极宽足彩量化分析软件），是我国首个开源的足彩量化回溯分析系统，内置 TOP-AI 极宽人工智能、机器学习模块库。
- 2017 年，发布我国首个足彩开源大数据项目 tfbDat2017 版，收录近 7 万场比赛数据、二百多万条赔率数据。

1.5 大数据图灵（足彩）原则

足球比赛的结果，从数学角度而言，是最简单的三选一，即胜、负、平。而真正的大数据分析，是在成千上万种可能中，选择最接近的进行匹配。连最简单的三选一都搞不定，来谈难度、复杂度高数百倍、上千倍的 n 选一（ $n>1000$ ），是不是有些不靠谱？

2014 年世界杯对于大数据、人工智能来说是个分水岭，这一年是人工智能的元年。微软、谷歌、百度等公司都有世界杯相关的项目。

有网友认为：这个标准，是高到永远不可能实现的标准！

原因很简单，真做到了，你不只是发财了，而且是彻底推翻了从帕斯卡开始的无数超级头脑和严密逻辑得出的概率论这一门数学理论！推翻一门理论在科学上屡见不鲜，但在数学上还未发生过。

这个标准并非高不可攀。

图灵测试，并非要求人工智能达到爱因斯坦的 IQ 才算合格，目前，人工大脑有小学生的水平就已经是最顶尖的了。

目前，人工智能最多相当 3~5 岁的儿童，基于人工智能的商业智能，和建立在二者之上的大数据，也只有 3~5 岁的智能，所以说大数据只是概念产品阶段。

足彩是最简单的三选一模式，随机盲选的概率都超过 30%，如果一个大数据分析模型连随机概率都无法超过，那么只能说这个模型不靠谱。

1.6 主要在线彩票资源

目前，我国正式的彩票网站有近百家，其中大部分是上市企业或者上市企业的关联公司创办的，这些网站都可以提供足彩及其他各种彩票的信息数据。

这些在线资源，大体可以分为以下这些类型。

- 热门彩票网站。
- 热门足彩彩票网站。
- 热门彩票论坛社区。
- AI 智能足彩预测网站。
- 传统足彩预测机构。
- 热门彩票博客。
- 热门足彩推荐专家。
- 热门足彩合买名人。

因为彩票产业在我国属于特种监管行业，经常进行整顿，特别是网络彩票，因此本节介绍的网络资源与实际情况可能会有所差异。

下面具体介绍部分相关的网络资源。

1. 热门彩票网站

- <http://www.zhew.com>，中彩网。
- <http://www.lottery.gov.cn>，体彩网。
- <http://lottery.sina.com.cn>，新浪彩票。
- <http://caipiao.sohu.com>，搜狐彩票。
- <http://sports.qq.com/lottery>，QQ 彩票。
- <http://www.china-lottery.net>，中华彩票。
- <http://www.55125.cn>，中国彩吧。
- <http://www.17500.cn>，乐彩网。
- <http://www.16788.cn>，天吉网。

- <http://www.cz89.com>, 牛彩网。
- <http://www.800820.net>, 天齐网。
- <http://www.55128.cn>, 彩吧助手。
- <http://www.fcp.cn>, 777 福彩。
- <http://www.55126.cn>, 彩搜网。
- <http://www.8200.cn>, 彩宝网。
- <http://www.ssqzj.com>, 3D 之家。
- <http://www.scw98.com>, 南方双彩。
- <http://www.cjcp.com.cn>, 彩经网。
- <http://www.china-lottery.net>, 公益时报。
- <http://www.starlott.com>, 星彩网。
- <http://www.97654.com>, 好运彩。
- <http://www.fa818.com>, 东彩网。
- <http://www.170win.cn>, 170 彩票网。
- <http://www.fenbaihe.com>, 粉百合。
- <http://www.cncaiba.cn>, 新彩吧。
- <http://www.55127.cn>, 7 彩网。
- <http://www.bcyz777.com/bbs/forum.php>, 博彩 E 族。
- <http://www.cocololo.com>, 彩之网。
- <http://www.newcp.cn>, 新彩网。
- <http://www.pinble.com>, 拼搏在线。
- <http://www.cpyjy.com>, 彩票研究院。
- <http://www.taocai.cn>, 淘彩网。
- <http://www.haocw.com>, 好彩网。
- <http://www.17mcp.com>, 众彩网。
- <http://www.51985.net>, 彩民之家。
- <http://www.027cp.com>, 搏彩王。
- <http://www.zibocn.com>, 智博网。
- <http://www.dddcun.com>, 3D 村。
- <http://www.gtjd.net>, 3D 基础。
- <http://www.cpew.cn>, 彩易网。

- <http://www.cp2y.com>, 彩票 2 元网。
- <http://www.3216.cc>, 彩博士。
- <http://www.hcw888.com>, 好彩 888。
- <http://www.50cp.com>, 50 彩票网。
- <http://www.sdps365.cn>, 365 彩票网。
- <http://www.c393.com>, 393 彩票网。
- <http://www.smswriter.com>, 华彩网。
- <http://www.gq01.com>, 大星彩票。
- <http://www.cp121.com>, 彩票 121。
- <http://www.cpew.cn>, 彩易。
- <http://www.500.com>, 500WAN 彩票。
- <http://www.okooo.com>, 奥客彩票。
- <http://www.aicai.com>, 爱彩网。
- <http://www.lecai.com>, 乐彩。
- <http://caipiao.taobao.com>, 淘宝彩票。
- <http://caipiao.163.com>, 网易彩票。
- <http://www.cpdyl.com>, 大赢家。
- <http://www.cailele.com>, 彩乐乐。
- <http://www.iletou.com>, 爱乐透。
- <http://www.kuaicaile.com>, 快彩乐。
- <http://888.sports.qq.com>, QQ 彩票。
- <http://www.310win.com>, 彩客网。
- <http://www.ffcp.cn>, 非凡彩票。

2. 热门足彩彩票网站

- <http://www.sporttery.cn>, 中国竞彩网。
- <http://www.zucal310.com>, 足彩 310。
- <http://www.zgzcw.com>, 足彩网。
- <http://www.500wan.com>, 500WAN。
- <http://www.aibo123.com>, 爱波网。
- <http://www.jc258.cn>, 竞彩 258。

- <http://www.310win.com>, 彩客网。
- <http://sports.sina.com.cn/l/sporttery>, 新浪竞彩。
- <http://www.cpdj.com>, 大赢家。
- <http://www.bet007.com>, 探球网。
- <http://www.titan24.com>, 体坛网。

3. 热门彩票论坛社区

- <http://bbs.55125.cn>, 彩吧论坛。
- <http://bbs.52cp.com>, 双彩论坛。
- <http://bbs.zhew.com>, 中彩论坛。
- <http://bbs.16788.cn>, 天吉论坛。
- <http://bbs.17500.cn>, 乐彩论坛。
- <http://bbs.bwlc.net>, 北京福彩论坛。
- <http://bbs.cnhubei.com/forum-197-1.html>, 东湖福彩论坛。
- <http://bbs.hnticai.com>, 湖南体彩论坛。
- <http://bbs.17mcp.com>, 众彩论坛。
- <http://bbs.pinble.com>, 拼搏在线。
- <http://bbs.haocw.ocm>, 好彩论坛。
- <http://bbs.310win.com>, 彩客论坛。
- <http://bbs.cpdj.com>, 大赢家论坛。
- <http://bbs.500wan.com>, 500WAN 论坛。
- <http://bbs.okooo.com/forum.php>, 澳客论坛。
- <http://bbs.fa818.com>, 东北彩票论坛。
- <http://bbs.ai123.com>, 爱波论坛。
- <http://bbs.zgzcw.com>, 猜猜论坛。
- <http://bbs.aicai.com>, 爱彩论坛。
- <http://bbs.cai898.com>, 小财神论坛。
- <http://88.hinews.cn/bbs/forum-2-1.html>, 南国论坛。
- <http://www.33633.cn>, 天中网。

4. AI 智能足彩预测网站

对于 AI 智能足彩预测网站,除了传统的足彩网站,近年来出现了不少做大数据、

人工智能项目的企业，甚至许多做手机 App 软件的公司，也开始进入这个领域。

- 百度足球赛事预测——单场预测，<http://trends.baidu.com/football>。
- 足彩预测—足彩分析—足球魔方官网—欢呼吧财富社区，<http://www.huanhuba.com/>。
- 310DATA，<http://310data.wozhongla.com/index.html>。
- 彩球网—体育大数据应用，<http://www.caiqr.com/>。
- 数据购买—球探网，<http://guess1.win007.com/sale/>。

5. 传统足彩预测机构

传统足彩预测网站除了传统的彩票网站外，还有不少体育媒体，甚至资深的体育记者、博客参与。

- 竞彩足球专家推荐_数字彩票专家预测_彩票专家预测-中国足彩网，<http://cp.zgzcw.com/zjtj/index.jsp>。
- 足球专家&高手推荐_竞彩专家预测-中国足彩网，<http://cp.zgzcw.com/zjtj/jc/expertList.jsp>。
- 足彩八方预测—中国足彩网(注意每场 ID 不同)，<http://fenxi.zgzcw.com/2058369/bfyc>。
- 足彩八方预测—中国足彩网八方预测，<http://fenxi.zgzcw.com/>。
- 搜狐足彩预测，足彩专家推荐—搜狐，<http://caipiao.sohu.com/s2013/sohuyuce/>。
- 足彩分析预测—搜狐，<http://caipiao.sohu.com/sports/yuce/>。
- 足彩预测、足彩专家预测—彩客网 <http://www.310win.com/tag/zucaiyuce/>。
- 足彩专家预测_足球彩票预测_足彩推荐_足彩分析_大彩网，<http://www.dacai.com/news/zc/zjyc/>。
- 足彩媒体预测_足球彩票预测_足彩推荐_足彩分析_大彩网，<http://www.dacai.com/news/zc/mtyc/>。

1.7 主要在线足彩数据源

这几年大数据、云计算风生水起，开放的数据源 API 接口越来越多，也有不少机构提供了足彩等体育数据接口。此类数据源 API 从收费模式来看，分为收费 API 与免费 API 两种。

从行业背景来看，分为大数据综合 API 和专业体育数据 API。专业体育数据 API 价格非常昂贵，广大个人用户难以承受，在此，只简单介绍一些综合性的大数据 API 网站，如图 1-3 所示。



图 1-3 大数据 API 网站示例

常用的数据网站如下。

- 开发者数据，<http://www.haoservice.com/>。
- 聚合数据，<https://www.juhe.cn/>。
- 阿凡达数据，<http://www.avatardata.cn/>。
- 免费接口 API 汇集，<http://www.apifree.net/>。
- 91 查，<http://www.91cha.com/>。
- 云聚数据，<http://www.36wu.com/>。
- JSON API 免费接口，<http://www.bejson.com/knownjson/webInterface/>。
- 数据堂，<http://www.datatang.com/>。
- 数据堂-数+，API <http://www.apiinside.com/>。
- Baidu—API 集市，<http://apistore.baidu.com/>。

在这些综合性的大数据网站中，有关的体育数据、足彩数据、足球联赛数据经常调整，请读者在使用前专门下载相关的 API 文档和案例程序。

1.8 足彩基础知识

500 彩票网站等大型的彩票网站中都有相关的足彩和赔率基础知识介绍，为节省篇幅，本书对于这些基础知识不再重复介绍。

笔者推荐读者通过 500 彩票网站的 500 研究院，如图 1-4 所示，系统学习足彩的基本知识。

500 研究院网址：<http://zx.500.com/zhuanti/school/>。



图 1-4 500 研究院网页截图

1.9 学习路线图

本书是“零起点 Python”系列图书，要更好地学习本书，掌握相关的配套程序，最好具备以下基础。

- Python 编程基础，不懂 Python 语言的读者，先用一周时间学习 Python 基本知识。
- 掌握足彩赔率等博彩基础知识，参见上一节的内容。
- 花几天时间学习 Pandas（潘达思）数据分析软件基础操作。

足彩大数据分析与金融量化分析类似，都属于大数据的范畴，有关的学习路线图、参考资料、图书也大体相似。

其中差异较大的是行业背景知识，量化强调的是金融知识，足彩知识相对简单很多，大型的彩票网站都有相关的基础知识介绍。

Top 极宽量化社区有“Python 量化与 zwQuant 学习路线图”，读者可以参考，网址是 <http://topquant.vip/forum.php?mod=viewthread&tid=6&extra=page%3D1>。

2

第 2 章

开发环境

2.1 数据分析首选Python

2.1.1 大数据，why Python

足彩与金融量化类似，都属于数据分析领域，而数据分析，笔者首选的编程工具就是 Python 语言。

在与网友的互动中，经常有网友认为自己投资失败或者量化策略失效，是因为 Python 开发环境不好，希望能重新设计一个，甚至希望用 C、Java 来重新开发量化软件。

这种观点是绝对错误的，对于这类问题，笔者曾经这样答复：

“这里面有个误区，量化的核心是策略，至于量化软件的运行速度、用户 GUI 界面等是程序员的事情；大家是做金融的、是操盘，会写策略就好了。就像会计，你觉得 Excel、用友软件或金蝶软件的某些功能不好用，另外再选一套财务软件就是了，完全没必要自己也去写一套财务软件。”

在软件工程领域，有一句名言“Don't Reinvent the Wheel”（不要重复发明轮子），这句话说得非常经典，也非常有道理。

至于为什么选择 Python 语言作为数据分析和量化分析的首选开发平台，读者可以参考极宽 Top 量化社区的介绍：“zwPython 史前故事”（<http://ziwang.com/forum.php?mod=viewthread&tid=61>）。

在“zwPython 史前故事”中，这样写到：

伴随《零起点 Python 大数据与量化交易》一书的出版，以及 zwQaunt 量化开源软件、zw-dat 开源金融大数据项目的成功与普及，其背后的功臣 zwPython 集成式 Python 开发平台，也逐渐被广大用户认可与接受。

今天，恰好是 2016 年圣诞节，回顾几年前的艰难抉择，非常庆幸选择了 Python 语言，选择了成功。

2012—2013 年是 R 语言最辉煌的时候，也是 Python2 版本向 Python3 版本过渡的艰难时刻，真可谓是青黄不接。

那时，新一代数据分析神器 Pandas（潘达思），以及 Python 现代风格的数据可视化模块绘图 plotly、seaborn 都还在萌芽之中。

在当年的博文“2013 年字库产业感悟及资料书单”（http://blog.sina.com.cn/s/blog_7100d4220101dtvz.html）中，笔者曾经写道：

字王新一代智能字模 sdk（zwPython 前身），准备采用 1k~2k 像素点阵的字模，数据处理量是 256 点阵的 10~100 倍，目前正在做前期规划，大体有以下三个框架。

- MATLAB，无疑是首选，支持 CUDA 加速，处理速度可以提升 10~200 倍，可以达到超算中心小型机的速度，可惜软件太大，2013 新版本有 2 张 DVD，安装耗时，且正版价格太贵。
- R 语言，是如今大数据的热选，是统计行业、精算师的 Photoshop，今年最小的 V3 版本才 50MB，而且是免费开源软件，图像支持也不错，正在考察中。
- Python 无所不在，也是开源软件，而且有 PythonXY，是专业的 Python 图像、数据处理计算整合包。开源的字库设计软件 FontForge，其主体模块也是采用 Python 编写的。问题是，Python 目前正处于 2.0 到 3.0 版本的大变迁时代，Python 3.0 语法虽然变动不多，但部分语句不兼容 2.0 版本，这也是个大问题。PythonXY 目前貌似采用的还是 2.x 版本，因为 PythonXY 集成的第三方数学模块太多了，全部升级到 3.0 版，估计至少要 3~5 年。

统计学与量化分析的关系本质上就是统计学与数据分析的关系，笔者当时在博文中这样写道：

字库设计与统计学从表面上看毫无关系。

其实不然，数字化后的字模是 100% 的纯数据处理，而数据处理，正是统计学的领域，也是 R 语言的“拿手好戏”。

以前字王的字模 sdk，关于字型轮廓的间距模块，只涉及到最短路径和绝对距离两种类型算法。

这次看过 R 语言才发现，统计行业的间距模块种类甚多，而且都有现成的模块，如基本距离、绝对距离、曼哈顿距离、欧氏距离、明氏距离、马氏距离和二值定性距离。

在《零起点 Python 大数据与量化交易》一书的前言中，笔者也写到：

2014 年，高盛、大摩的雅典娜和黑石计划都不约而同地选择了 Python 作为金融行业的标准编程语言。

资本的力量是强大的，也是冷酷无情的。

2016 年 5 月，《华尔街日报》报道，目前华尔街的三大编程语言是：C、Java 与 Python。其中，C 与 Java 成为三大语言之一有两方面原因：一方面是由于历史积累，另一方面是系统架构设计的需要。而在应用领域 Python 更胜一筹，因为 Python 已经成为金融行业量化领域的标准编程语言。

回顾几年前的艰难抉择，笔者再一次庆幸当时选择了 Python，选择了成功。

正是因为经历了这样一个艰难曲折的过程，所以笔者对于软件工程的这句名言“Don't Reinvent the Wheel”（不要重复发明轮子）更加深有体会。

足彩和量化分析一样，本质上都是数据分析，如果读者有兴趣，还可以浏览笔者的博客《大数据，why Python》，网址是 <http://ziwang.com/forum.php?mod=viewthread&tid=58&extra=page%3D1>。

2.1.2 入门简单，功能强大

Python 是最适合初学者学习的编程语言之一，也是目前 IT 行业唯一的入门简单、功能强大的工业级开发平台。

事实上，目前 Python 已经超越普通编程语言，几乎成为 IT 行业的万能开发平台。

1. 入门简单

任何熟悉 JavaScript 脚本、VB、C 语言、Delphi 的用户，通常一天即可学会 Python。

即使是不会编程的设计师、打字员，一周内也能熟练掌握 Python，其学习难度绝对不会高于 Photoshop、五笔，至少笔者现在还不会使用五笔字型。

2. 功能强大

海量级的 Python 模块库提供了 IT 行业最前沿的开发功能。

- 大数据方面，Pandas 已经逐步碾压 R 语言。
- CUDA 编程方面，Python 与 C (C++)、Fortran 是 NV 官方认可的三种编程语言，也是目前唯一适合 PC 平台的 CUDA 编程工具。
- 机器学习方面，TensorFlow、PyTorch、Scikit-learn、Theano 是国际上最热门的机器学习平台。
- 自然语言方面，NLTK 是全球首选的自然语言处理平台；spaCy 是工业级 NLP 平台。
- 人脸识别方面，OpenCV 有光流算法、图像匹配和人脸算法，使用 Python 简单而且优雅。
- 游戏开发方面，Pygame 提供图像、音频、视频、手柄、AI 等全套游戏开发模块库。
- 字体设计方面，FontForge 是唯一商业级的字体设计开源软件，内置脚本和底层核心的 Fonttools，都是使用 Python 开发的。
- 电脑设计方面，Blend、GIMP、Inkscape、Maya、3ds Max 都内置或扩展了 Python 语言支持。

上述 Pandas、CUDA、TensorFlow、PyTorch、Scikit-learn、Theano 均为 Python 模块库或 IT 行业术语。

目前热门的 iOS、安卓、WP 等手机 App 应用开发，也可以用 Python 开发，但基本都是商业收费模块，因此未集成到 zwPython 软件包，读者可以自行查询。

吉多·范罗苏姆 (Guido van Rossum) 是一名荷兰计算机程序员，他作为 Python 程序设计语言的作者而为人们所熟知。

他为 Python 设计的目标是：

- 一门简单、直观的语言并与主要竞争者一样强大；
- 开源，以便任何人都可以为它做贡献；
- 代码像纯英语那样容易理解；
- 适用于短期开发的日常任务。

既然 Python 如此美好，并且是 100% 免费的开源软件，学习 Python 的人也越来

越多,那么为什么 Python 始终还只是一种小众语言呢(相对 VB、C、C#、JavaScript)?

笔者认为,Python 的“大众化”之路,存在以下两个瓶颈。

- 配置,软件行业有句俗话,“搞懂了软件配置,就学会了一半”。对于 Python、Linux 及许多开源项目而言,80%的问题都出现在配置方面,尤其是模块库的配置。
- OOP (面向对象程序设计),大部分人都认为 Python 是一种“面向对象”的编程语言,而 OOP 的编程风格,业界公认比较繁杂。

如果能够解决好以上两个问题,那么 Python 的学习难度可以降低 90%,而应用领域和开发效能可以瞬间提升 10 倍,而且这种提升是零成本的。

2.1.3 难度降低 90%, 性能提高 10 倍

为此,笔者在 winPython 软件包的基础上,推出了“zwPython”——字王集成式 Python 开发平台。

- 业界首次提出“零配置、零对象”研发理念,绿色软件封装模式,类似 Mac 开箱即用风格,无需安装,解压即可直接使用,还可以放入 U 盘,支持 Mob-App 移动式开发编程。
- “外挂”式“核弹”级开发功能,内置很多功能强大、IT 前沿的开发模块库,例如 OpenCV 视觉/人脸识别、CUDA 高性能 GPU 并行计算 (OpenCL)、Pandas 大数据分析、TensorFlow、PyTorch 机器学习、NLTK 自然语言处理。
- 便于扩展,用户可以轻松增删相关模块库,全程智能配置,无需用户干预,就好像复制文件一样简单,而且支持 U 盘移动便携模式,真正实现了“一次安装,随处可用”。
- 针对中文开发文档缺乏、零散的问题,内置多部中文版 OpenCV、FontForge 和 Python 入门教材。
- 内置多款中文开源 Truetype 字库。
- 大量示例脚本源码,涵盖 OpenCV、CUDA/OpenCL、Pygame 等。

如此种种,只是为了便于 IT 行业外的用户能够零起步、快速入门,并且在短时间内能够应用到生产环节中。

ZwPython 的前身是 zw2015sdk,即字王智能字模设计平台,原设计目标是为广大设计师提供一款统一的、可编程的字体设计平台,便于大家交流。

设计师、美工都是文艺青年，所以简单是必须的，开箱即用也必须是标配。

- 用来做设计，图像处理 PIL 和 Matplotlib 模块是必须的。
- 集成了 OpenCV 作为图像处理、匹配模块，自然也提供了机器学习功能。
- 字模处理数据量很大，属于大数据范畴，必须集成 Scipy、Numpy 及 Pandas 数据分析模块。
- 由于原生 Python 速度慢，所以增加了 PyCUDA、OpenCL 高性能 GPU 计算模块。

如此等等，一而再、再而三地扩充，发现 zwPython 已经基本覆盖了目前 Python 和 IT 编程领域 90% 的应用，因此又增加了部分模块，将 zwPython 扩展成为一个通用的、集成式 Python 开发平台。

2.1.4 “零对象”编程模式

虽然很多人都认为 Python 是一种“面向对象”的编程语言，但对初学者而言，把 Python 视为一种 BASIC 风格的、过程式入门语言，学习难度可以降低 90%，基本上学习一小时即可动手编写学习代码。

有人说，“面向对象”最大的好处是方便把人脑子搅乱。

Windows、Linux、UNIX、Mac OS X 内核都是 C 语言、汇编语言写的。有一种系统是 C++ 语言写的内核，就是诺基亚的塞班系统，现在已经“死”掉了，据说代码量比 Windows XP 还大，连他们自己的程序员都无法维护。

零对象编程模式，就是用 BASIC 的方式学习 Python。

这只是笔者向 Python 等编程语言的入门用户，提出的一种全新的学习理论，一家之言，仅供参考。

“零配置”很容易理解，关于“零对象”，下面再补充几点。

- 不写“面向对象”风格的代码，不等于不能使用，对于各种采用“对象”模式开发的模块库，我们仍然可以直接调用。
- 将 Python 视为非“面向对象”语言，并非“大逆不道”，事实上，许多人认为 Python 也是一种类似 lisp 的“函数”编程语言。
- 笔者编程十多年，从未用过“面向对象”模式，编写过一行“class”（类对象）代码，依然可以应对各种编程工作。
- “面向对象”编程理论目前在业界仍然争论不休，入门者功力不够，最好避开强

者间的火力杀伤。

- “面向对象”的鼻祖 C++ 11 标准，直到 2015 年，依然处于推广阶段，而且争议纷纷。
- “面向对象”过于复杂，与 Python 的优雅风格天生不合。

2.2 用户运行平台

本节主要讲解 Python 开发环境和数据包的配置、应用流程方面的知识。

本书所有案例程序均采用纯 Python 语言开发，除特别指明外，默认使用 Python 3 语法，均经过 zwPython 平台测试。

zwPython 是极宽公司推出的一个 Python 集成版本，功能强大，是国内免费开源软件，其用户手册封面如图 2-1 所示。系统内置了数百种专业的 Python 模块库，无需安装，解压即用。有关 zwPython 的使用，可参考软件自带的《zwPython 用户手册》。

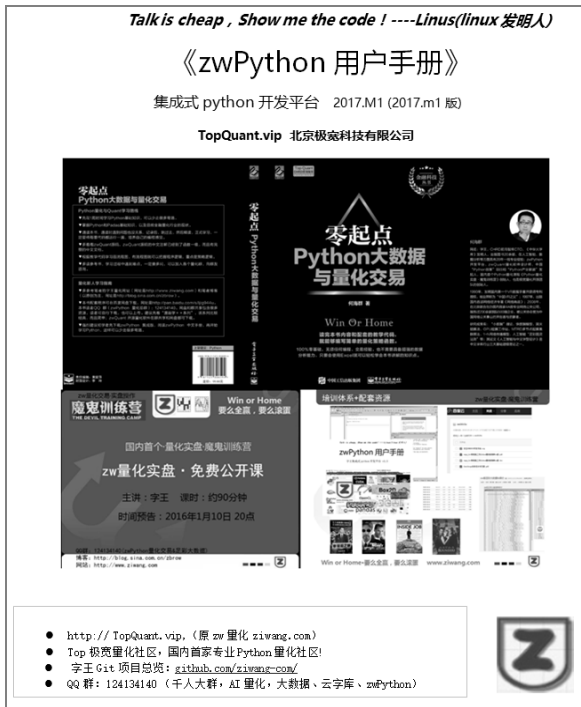


图 2-1 《zwPython 用户手册》封面

本书所有案例程序可用于 zwPython 平台及各种支持 Python 3 的设备平台,包括 Linux 操作系统、Mac 苹果电脑,以及安卓系统,甚至树莓派。

其他非 zwPython 用户运行本书程序,如果出现问题,通常是缺少有关的 Python 模块库,可以根据调试信息安装相关的 Python 模块库,再运行相关程序。

限于篇幅,关于 Python 语言和 Pandas (潘达思) 数据分析软件的基本操作,请读者查看有关图书。

初学者最好下载足彩教学版,下载地址见 Top 极宽量化社区的“下载中心”,网址是 <http://topquant.vip> 或 <http://ziwang.com>。

为方便读者学习使用 Python 足彩大数据,笔者特意将和初学者有关的教学资源打包成一个独立的压缩包,便于使用。

- 足彩教学版为一个独立压缩包,解压即可。
- 足彩教学版内置了 zwPython 开发平台 (python 3.5 版本)、tfbDat 开源足彩大数据包、TopFootball 极宽足彩量化分析系统和 zc_demo 配套量化教学课件程序。

2.3 程序目录结构

本书配套程序的工作目录是 zwPython\zc_demo,这个目录也是默认的工作目录,凡是没有标注目录的脚本文件,一般都位于该目录。

有关的程序会定时在读者群发布更新,请读者及时下载。

相比普通的 Python 版本,教学版的 zwPython 目录中多了一个 zc_demo 目录。

zc_demo 目录收录的是相关培训课程的配套代码和所需数据,zc_demo 目录也可以复制到其他目录,建议放到 zwPython 根目录下。

zwPython 目录结构中的其他子目录如下。

- \zwPython\doc\ : 用户文档中心,包括用户手册和部分中文版的模块库资料。
- \zwPython\py35\ : Python 3.5 版本系统目录,除增加、删除模块库外,一般不需要改动本目录下的文件,以免出错。另外,如果日后 Python 版本升级,这个目录也会变化,如 Python 3.6,会采用 py36 的目录。
- \zwPython\demo\ : 示例脚本源码。
- \zwPython\zwrk\ : zw 工作目录,用户编写的脚本代码文件建议放在本目录下。
- \zwPython\TopFootball\ : TopFootball 又称 Top Quant for Football (简称 TFB),

极宽足彩量化分析系统。

极宽足彩量化分析系统与 zwPython 进行了集成处理，可直接 import 使用，支持 Python 3.x。

移植时或使用其他 Python 环境时，可以把 TopFootball 目录下的脚本文件全部复制到自己的代码工作目录，注意代码里面有关数据文件目录的设置。

2.4 tfbDat足彩数据包

tfbDat 足彩数据包是 top football Data 的缩写，是原 zcDat 的升级版，是国内首个开源的足彩大数据项目，目前共收录了 2010—2017 年近七万场比赛数据和二百多万条足彩赔率数据。

tfbDat 足彩数据包目录结构如图 2-2 所示。

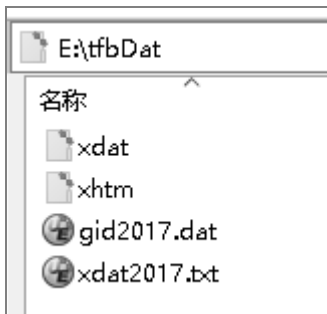


图 2-2 tfbDat 足彩数据包目录结构

其中，具体目录结构如下。

- \xdat\：赔率数据，目前只收录了欧赔数据。
- \xhtm\：网页文档。
 - \xhtm\ghhtm\：gid 比赛数据网页。
 - \xhtm\htm_oz\：欧洲赔率数据网页。
 - \xhtm\htm_az\：亚洲赔率数据网页（暂缺）。
 - \xhtm\htm_sj\：分析数据网页（暂缺）。
- gid2017.dat，比赛基本数据，共收录近七万场比赛数据。
- xdat2017.txt，赔率数据合集，共收录近七万场比赛的二百多万条赔率数据。

2.5 Spyder编辑器界面设置

2.5.1 开发环境界面设置

在设置界面之前，可随意把一个 Python 源码文件，用鼠标拖到 Python 编程语言编辑器 Spyder 的编辑框中，如图 2-3 所示。

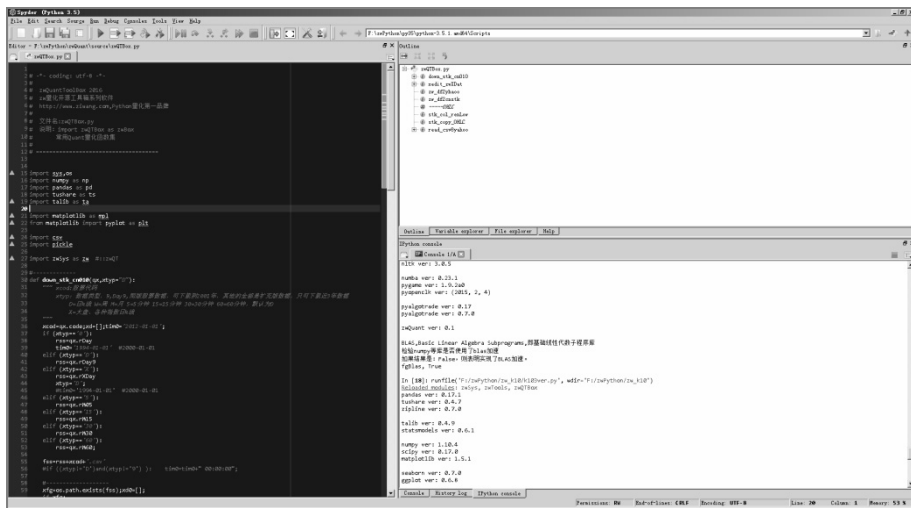


图 2-3 Python 编程语言编辑器 Spyder 编辑框界面

Spyder 编辑器的界面设计非常合理，参考了 MATLAB，特别适合量化分析。国际投行一般都选择这种布局作为标配。

通常需要优化的只有 Outline（导航）面板，又称函数列表面板，类似 Delphi 语言的 Struct 函数列表面板。

在 Spyder 编辑器默认配置中，Outline 面板是不显示的，单击菜单“View→Panels→Outlines”，如图 2-4 所示，将显示 Outline 面板。

Outline 面板显示后，它的默认位置是在代码编辑器和右侧窗口的中间。

建议单击 Outline 面板左上角的“窗口缩放”按钮，拖动面板到右上方，将其与 Var（变量）面板、File（文件）面板等合并。

Outline 面板的作用是对代码中的函数、类、变量进行快速导航定位。单击 Outline

面板的函数、类、变量名称后，左侧代码编辑器就会自动移动到相关代码，如图 2-5 所示。对于大型项目而言，使用 Outline 面板可以提高效率。

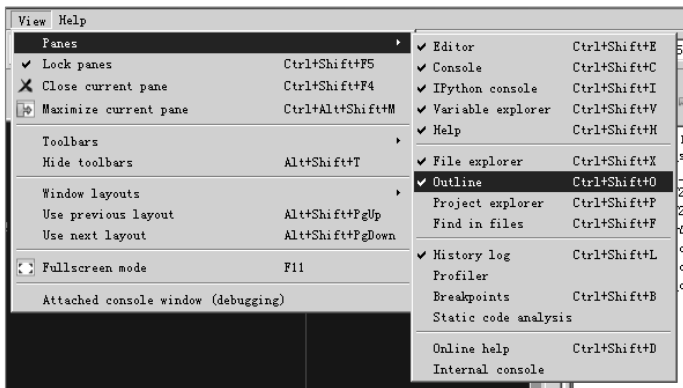


图 2-4 显示 Outline 面板操作



图 2-5 Spyder 编辑器 Outline 面板

需要注意的是，由于 Spyder 未来版本将升级，具体操作界面和细节可能会有所不同，本书其他软件和模块也类以，这属于正常情况，读者无需担心。

2.5.2 代码配色技巧

zwPython 的 IDE 代码编辑器是 Spyder，默认配色是 Spyder 模式，采用白底黑字，与传统的 IDE 环境差别很大，如图 2-6 所示。

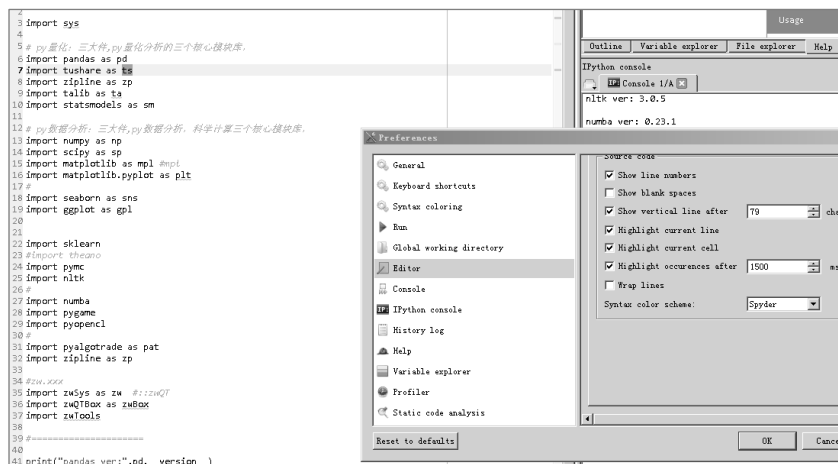


图 2-6 Spyder 编辑器配色模式

如图 2-7 所示是最新的 Delphi-xe10 的编辑器配色模式（Twilight 模式）。

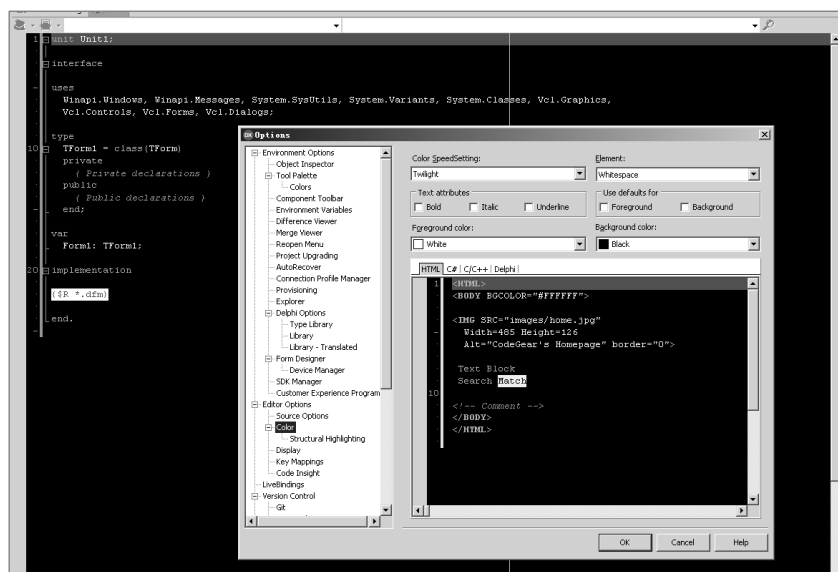


图 2-7 Delphi-xe10 编辑器配色模式

这种黑底模式也是微软等开发平台标准的代码编辑器配色模式。幸运的是，Spyder 编辑器内置的配色模式里也有类似的模式。

运行 Spyder 编辑器，单击菜单“Tools→Preferences”，打开“Preferences”对话

框。在左侧的列表框中选择 Editor(编辑器), 在右侧 Display(显示)选项卡的“Syntax color scheme”(语法配色方案)下拉列表框中选择“Spyder/Dark”(暗调)模式即可, 如图 2-8 所示。

不同版本的 Spyder 编辑器调整细节会有所不同, 请读者注意。

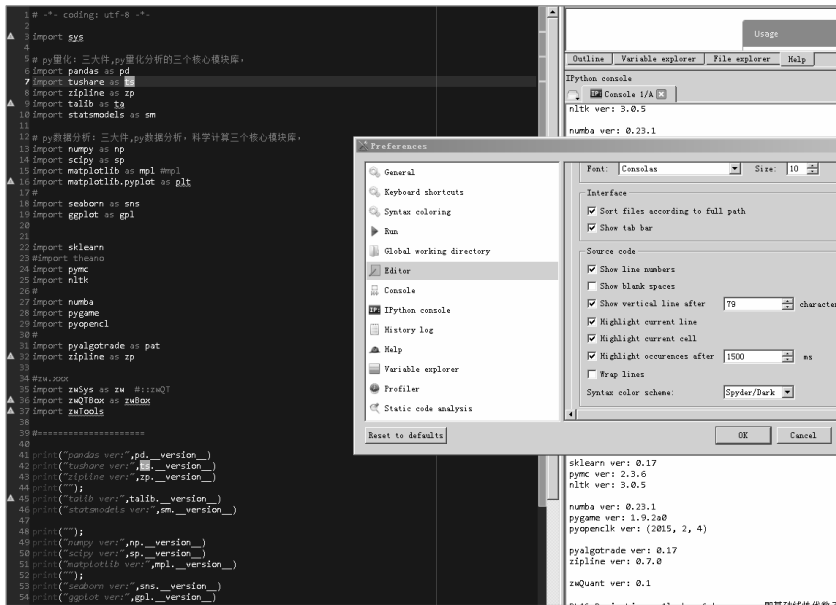


图 2-8 调整 Spyder 编辑器配色模式

2.5.3 图像显示配置

Python 语言的 Spyder 编辑器默认的图像显示尺寸对于高清显示器来说尺寸有些小, 需要进行调整, 具体步骤如下。

- (1) 单击菜单“Tools→Preferences”, 打开“Preferences”对话框。
- (2) 单击左侧列表框中的“IPython console”(IPython 控制台)。
- (3) 在对话框的右侧, 选择“Graphics”选项卡。
- (4) 在“Graphics backend”选项区中, “Backend”选项默认为“Inline”, 一般不需要修改。如果要进行交互分析, 可以设置为“Automatic”(自动模式)或者“Qt”(Qt 模式)。

(5) 在“Inline backend”选项区中可以调整内置图像的大小，默认值“Width”为 8、“Height”为 5，建议改为“Width”为 10、“Height”为 6。

此外，建议勾选对话框上部的“Automatically load Pylab and NumPy modules”复选框，会自动加载 Pylab、NumPy 模块，如图 2-9 所示。

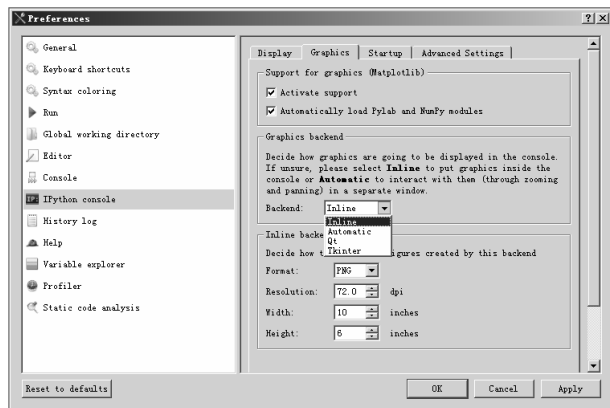


图 2-9 调整 Spyder 编辑器图像显示尺寸

2.5.4 重剑无锋

对于数据分析、量化分析的开发平台，笔者的一贯主张就是直接使用 zwPython 内置的 Spyder 开发平台。

Spyder 的工作界面，经过多年一线数据分析、量化分析实盘操作人员的反馈调整和设计优化，对于数据分析、金融量化工作者而言，已经是一种非常理想的工作界面，也可以说是最好的工作界面，具体理由如下。

- Spyder（前身是 Pydee），是一个强大的交互式 Python 语言开发环境，提供高级的代码编辑、交互测试、调试等特性，支持 Windows、Linux 和 Mac OS X 系统。
- Spyder 最早发布于 2009 年，经过多年的升级优化，目前已经非常成熟，最大程度上减少各种 bug 对于实盘操作的干扰。
- Spyder 默认界面布局如图 2-10 所示，类似 MATLAB，集中了代码编辑、项目管理、变量检查与图形查看等多种功能，这种界面布局也是金融工程、数据分析行业的标准工作界面。

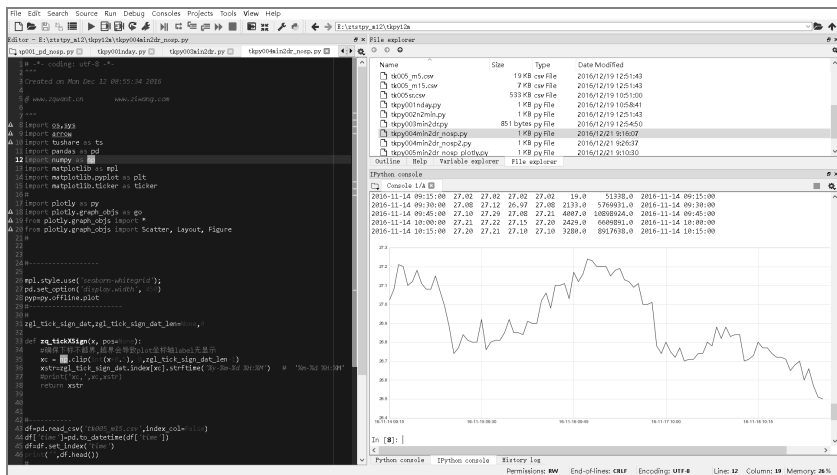


图 2-10 Spyder 工作界面

GUI 用户界面其实类似计算机的机箱，虽然华丽，但只是表层的东西，绝非核心因素。其实许多服务器采用的 Linux 操作系统，为了追求极致的性能，还在使用传统的纯文本界面，根本没有所谓的 GUI 用户接口。

笔者在设计极宽量化软件时，无论是最初的 zwQuant，还是后来推出的 zQuant-core（内核版），都是强调简单实用，这些案例也可以说是“kiss 法则”（保持简单）在软件工程中的具体应用。

也许，读者觉得 Spyder 界面过于朴素，这种朴素其实源自开源的历史与传承，可以看一些著名开源项目的网站，网页都非常简单朴素，有些甚至还是互联网起步阶段的文本模式。

- <http://www.apache.org>，Apache 开源项目网站。
- <https://github.com>，GitHub 开源项目网站。
- <http://www.lfd.uci.edu/~gohlke/Pythonlibs>，LFD 二进制 Python 模块库。
- <http://mirrors.163.com/>，网易开源镜像网站。
- <http://mirrors.sohu.com/>，搜狐开源镜像网站。

幸运的是，如今很多成功的互联网企业，如谷歌、百度也继承了这种朴素的传统，其搜索引擎的首页都是大片的空白，类似中国传统书法的留空，只有简简单单的搜索框。

在这种朴素的背后，是一种“重剑无锋”的体现。

最后，笔者再一次强调，量化分析的核心是策略，不是交易接口，不是自动下单，不是用户界面，更不是软件开发平台。

目前，Python 语言已经是数据分析、量化分析的行业标准编程语言，读者无需争议。

在试图质疑这些问题的时候，请重新审视一下软件工程的名言：“Don't Reinvent the Wheel”（不要重复发明轮子）。

2.6 Notebook 模式

zwPython 内置的 Notebook 支持模式，目前已经是 Python 源码交流的常用模式。事实上，Notebook 已经是数据分析、信息分享的 Web 标准模式。

Notebook 模式文件后缀名是.ipynb，类似 IE 的 MHT 网页打包格式，支持文字格式、排版、图像。

Notebook 模式运行方法如下。

- (1) 进入 py35 目录。
- (2) 运行 Jupyter Notebook.exe 程序。

Jupyter Notebook.exe 程序类似单机的本地 Web 服务器软件。

运行后，会自动调用默认浏览器，并访问默认网址 <http://localhost:8888/tree>，如图 2-11 所示。

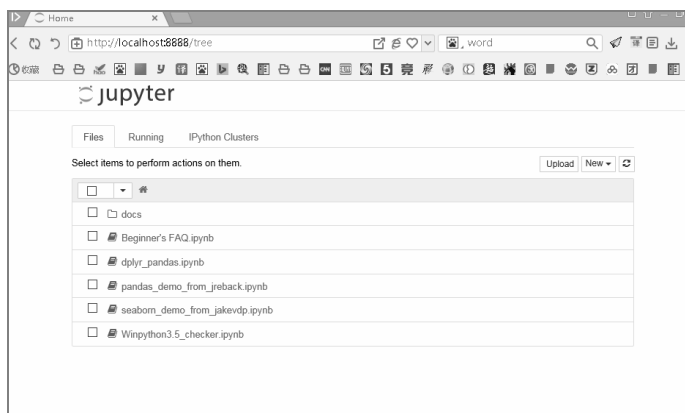


图 2-11 Notebook 模式

.ipynb 格式文件使用方法如下。

- (1) 运行 Jupyter Notebook.exe 程序，进入 Notebook 模式。
- (2) 单击右上角的“Upload”按钮查找文件，再单击对话框中的“upload”按钮，即可上传文件，或者用鼠标直接拖曳.ipynb 格式文件到浏览器窗口。
- (3) 上传文件后，单击相应的文件名，即可看到相应的脚本内容，以及运行结果和图片。

具体效果如图 2-12 所示，根据文件内容不同，其显示会有所不同。

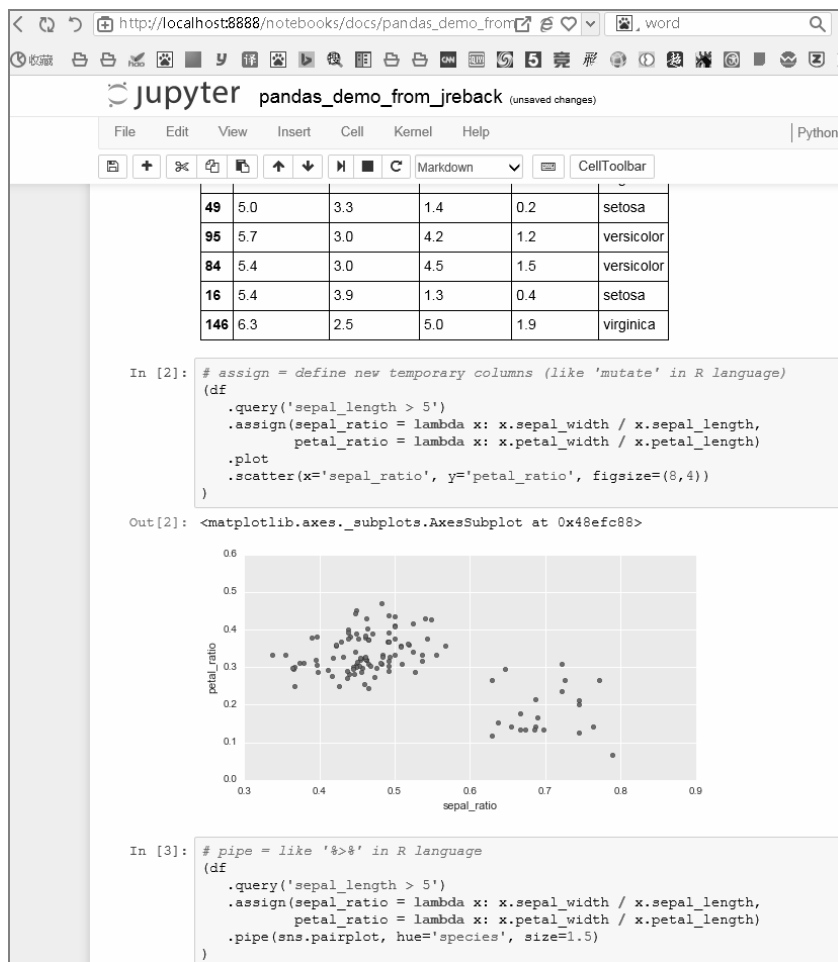


图 2-12 Notebook 模式运行效果

2.7 模块库控制面板

Python 的强大和方便，除了体现在海量的内置模块外，还体现在绿色版本、灵活方便的模块库管理功能上。

zwPython 模块库管理直接使用 winPython 的控制面板程序：WinPython Control Panel.exe。

控制面板程序 WinPython Control Panel.exe 位于 py35 目录下，不同版本位置不同，不能混用，请读者注意。

运行后界面如图 2-13 所示。

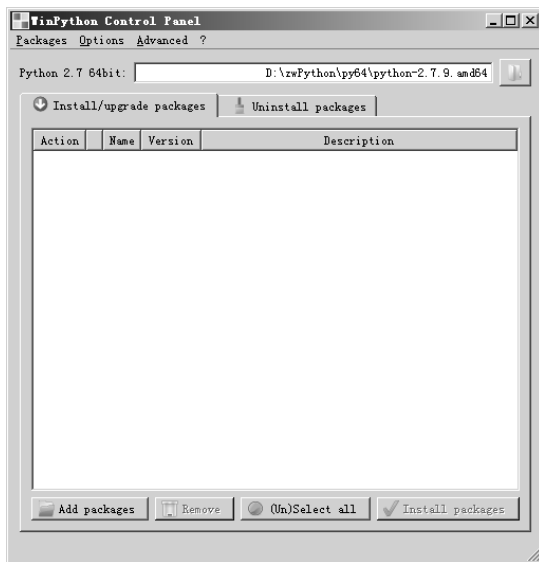


图 2-13 控制面板程序运行界面

zwPython 模块库的安装流程如下。

(1) 把下载的 Python 模块库复制到任意一个目录。

(2) 单击左下角的“Add packages”按钮，从模块目录选择模块文件名，即可完成模块库的添加。一次可选择添加多个模块库文件，如果模块库版本不对，会出现在提示对话框中，显示相关的出错模块名称；zwPython 系统是基于 64 位 python 3.x 版的，因此下载模块请选择对应的版本。

(3) 添加完毕后，单击右下角的“Install packages”按钮即可完成模块库的安装。

安装时，需要注意以下事项。

- (1) 模块安装完成后，可以删除相关的模块文件，不影响程序使用。
- (2) 多个模块安装时，每次最好不要超过 20 个，以免出错。

2.7.1 模块库资源

zwPython 模块库资源主要来自四个方面。

- 各大网络 Python 社区，主要是.zip、.gz 格式。
- PyPI (Python Package Index)，Python 官方模块库，主要是.zip、.gz 格式。
- LFD，欧文加州大学的非官方 Python 集成模块库，主要是.exe、.whl 格式。
- GitHub，全球最大的程序资源网站，注意选择 Python 语言版本。

运行控制面板程序 WinPython Control Panel.exe 后，单击右下角的“Add packages”按钮，可以发现，系统支持多种格式的模块库安装，如.zip、.gz、.exe 和.whl。

zwPython 在模块库安装方面很强大，体现在：

- 支持多种格式，除官方的.zip、.gz 格式外，还支持 LFD 的.exe 和.whl 格式；
- 绿色安装，一次安装，随处运行，支持 U 盘便携式开发。

Python 官方模块库网址：<https://pypi.python.org/pypi>。

GitHub 网址：<https://github.org>。

LFD 非官方模块资源网址：<http://www.lfd.uci.edu/~gohlke/pythonlibs/>（LFD 采用集成方式打包，特别适合于 OpenCV、CUDA 等大型模块库安装）。

LFD 全称是“Laboratory for Fluorescence Dynamics, University of California, Irvine.”，即动力学实验室，加利福尼亚大学/欧文加州大学。

欧文加州大学（简称为 UCI 或 UC Irvine，又常被译为加州大学欧文分校）成立于 1965 年，是加州大学 10 个校区之一，位于美国加州欧文。

2.7.2 模块库维护更新

控制面板程序 WinPython Control Panel.exe 还提供了模块库的维护和升级功能。

- 在 Uninstall 面板中选择需要维护的模块。

- 单击菜单 “Option→Repair packages”，如图 2-14 所示。

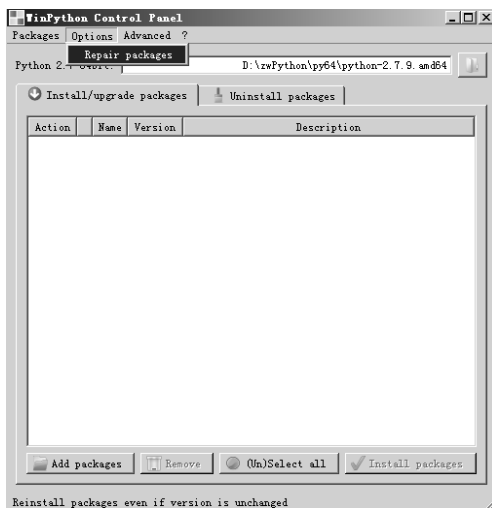


图 2-14 模块库维护

2.7.3 系统关联

控制面板程序 WinPython Control Panel.exe 还提供系统关联功能，操作步骤如下。

- 单击菜单 “Advanced → Register distribution”，如图 2-15 所示，即可将 zwPython 关联到 Windows 系统，关联后，可以直接在资源浏览器中运行 “.py” 脚本文件。另外，还可增加鼠标右键的.py 脚本文件与 spyder IDE 程序的关联编辑功能。
- 单击菜单 “Advanced → Unregister distribution”，即可解除关联。

通常，无需采用关联模式。



图 2-15 系统关联

2.8 使用pip命令更新模块库

有时，出于各种原因，使用控制台安装模块库会出现失败现象，或者需要批量更新模块库，这时就可以使用 pip 模块管理程序。

2.8.1 pip 常用命令

1. 列出已安装的包

```
pip freeze or pip list
```

2. 导出 requirements.txt

```
pip freeze > <目录>/requirements.txt
```

3. 在线安装<安装包>(模块库)

```
pip install <包名> 或 pip install -r requirements.txt
```

4. 指定版本

通过使用==、>=、<=、>、<来指定版本，不写则安装最新版本。

requirements.txt 内容格式为：

```
APScheduler==2.1.2
Django==1.5.4
MySQL-Connector-Python==2.0.1
MySQL-python==1.2.3
PIL==1.1.7
South==1.0.2
django-grappelli==2.6.3
django-pagination==1.0.7
```

5. 安装本地安装包

```
pip install <目录>/<文件名> 或 pip install --use-wheel --no-index
--find-links= wheelhouse/ <包名>
```

注意，<包名>前有空格。

可简写为：pip install --no-index -f=<目录>/<包名>。

6. 卸载包

```
pip uninstall <包名> 或 pip uninstall -r requirements.txt
```

7. 升级包

```
pip install -U <包名>
```

8. 升级 pip

```
pip install -U pip
```

9. 显示包所在的目录

```
pip show -f <包名>
```

10. 搜索包

```
pip search <搜索关键字>
```

11. 查询可升级的包

```
pip list -o
```

12. 下载包而不安装

```
pip install <包名> -d <目录> 或 pip install -d <目录> -r requirements.txt
```

13. 创建 wheel 格式的模块库

```
pip wheel <包名>
```

14. 指定镜像安装源

```
pip install <包名> -i http://pypi.v2ex.com/simple
```

国内 PYPI 镜像地址如下。

- pypi.v2ex.com/simple, V2EX。
- <http://pypi.douban.com/simple>, 豆瓣。
- <http://mirrors.aliyun.com/pypi/simple/>, 阿里云（推荐使用）。
- <http://pypi.mirrors.ustc.edu.cn/simple/>, 中国科学技术大学。
- <https://pypi.tuna.tsinghua.edu.cn/simple>, 清华大学。

- <http://pypi.hustunique.com/>，华中理工大学。
- <http://pypi.sdutlinux.org/>，山东理工大学。
- <http://pypi.mirrors.ustc.edu.cn/>，中国科学技术大学。
- <http://mirrors.sohu.com/python/>，搜狐镜像。

其他更多的有关 pip 的使用细节，读者可以自行查询。

2.8.2 进入 Python 命令行模式

Python 命令行模式与普通的命令行模式不同，因为集成了 Python 的运行环境参数。

许多新用户都是直接使用 Windows 或其他软件自带的 dos 命令进入 dos 命令行，然后运行 pip 命令，这样会出错。

正确的方法是，运行 py 目录下的 WinPython Command Prompt.exe 程序，如图 2-16 所示。

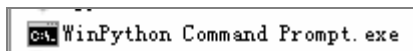


图 2-16 WinPython Command Prompt.exe 程序

- py27 版本: py27\WinPython Command Prompt.exe。
- py35 版本: py35\WinPython Command Prompt.exe。

运行后，会自动进入 Python 对应的子目录。

- py27 版本是 x:\zwPython\py27\python-2.7.10.amd64\。
- py35 版本进行了优化，目录是 x:\zwPython\py35\scripts\。

2.8.3 pip 安装模板

为了方便读者使用 pip 命令安装新的模块库，zwPython 集成了一个 pip01.bat 批命令模板，位于不同版本对应的目录下。

pip01.bat 批命令内容是：

```
rem tushare
pip install --upgrade tushare -i
https://pypi.tuna.tsinghua.edu.cn/simple
```

其中，tushare 是示例的模块库名称，可改为需要安装的模块库名称。

这个 pip01.bat 批命令会自动更新指定的模块库，如果找不到对应的模块，会重新安装。

因为 Python 官网速度很慢，所以在 pip01.bat 批命令中使用了国内的镜像源，如果出现网络问题，读者可以根据前面介绍的 PYPY 镜像站点更换对应的镜像网站。

2.8.4 pip 参数解释

pip 参数及其含义如表 2-1 所示。

表 2-1 pip 参数及其含义

参 数	参数含义
install	安装包
uninstall	卸载包
freeze	按一定格式输出已安装包列表
list	列出已安装包
show	显示包详细信息
search	搜索包，类似 yum 里的 search
wheel	按需求创建模块包
zip	不推荐，Zip individual packages
unzip	不推荐，Unzip individual packages
bundle	不推荐，Create pybundles
download	下载模块
hash	计算模块包的哈希数值
help	当前帮助
-h, --help	显示帮助
-v, --verbose	更多的输出，最多可以使用 3 次
-V, --version	显示版本信息后退出
-q, --quiet	最少的输出
--log-file <path>	以覆盖的方式记录 verbose 错误日志，默认文件
--log <path>	以不覆盖的方式记录 verbose 输出的日志

续表

参 数	参数含义
--retries <retries>	重试次数（默认 5 次）
--trusted-host <hostname>	可信任站点，不包括 https 站点
--timeout <sec>	连接超时时间（默认 15 秒）
--cert <path>	证书
--cache-dir <dir>	Cache 目录
--isolated	绝对模式，无视 Python 环境和用户设置
--upgrade	如果已安装就升级到最新版本

2.8.5 pip-install 参数选项

install 是最常用的 pip 参数，install 参数选项如表 2-2 所示。

表 2-2 install 参数及其含义

参 数	参数含义
-c, --constraint <file>	约束，使用给定的约束限制版本文件，此选项可多次使用
-e, --editable <path/url>	可编辑的<路径/网址>，在“开发模式”下安装一个项目
-r, --requirement <file>	按给定要求文件安装模块
-b, --build <dir>	建造模块包
-t, --target <dir>	目标目录
-d, --download <dir>	下载到目录
--src <dir>	用目录查看编辑项目
-U, --upgrade	所有的包升级到最新版本
--force-reinstall	升级时强制重新安装，即使它们已经是最新版本了
-I, --ignore-installed	忽略已经安装的模块包（与 Reinstalling 相反）
--no-deps	不安装依赖包
--install-option <options>	安装选项，使用 setup.py 的额外参数
--global-option <options>	全局选项
--egg	使用 eggs 模式安装，不用默认的 flat 模式
--root <dir>	根目录，安装使用的根目录
--prefix <dir>	安装前缀目录
--compile	编译 Py 文件为 Pyc 代码
--no-compile	不编译 Py 文件为 Pyc 代码

续表

参 数	参数含义
--no-use-wheel	不使用 wheel 模块包
--no-binary <format_control>	不使用二进制模块包
--only-binary <format_control>	不使用源码模块包，只用二进制模块包
--pre	包括预处理和开发版本
--no-clean	不清除创建的目录
--require-hashes	使用哈希验证

3

第 3 章

入门案例套餐

考虑到不少刚入门的读者对 Python 不甚了解，笔者在此特意增加了一章入门案例，通过几个简单的 Python 入门程序，帮助读者尽快掌握 Python 语言，熟悉开发环境。

3.1 案例3-1：第一次编程，“hello,ziwang”

软件安装完毕，我们就可以开始编写、运行 Python 脚本程序了。

(1) 单击工具栏中的“读取”按钮，打开“zc_demo\”目录下的 py301.py 脚本文件。

(2) 单击工具栏中部的绿色“运行”按钮“▶”。

程序很简单，只有一行代码：

```
print("hello,ziwang.com")
```

如图 3-1 所示为输出面板，运行后，在右下角的输出窗口可以看到“hello,ziwang.com”的字样，表示运行成功。

读者可以自己修改程序中引号里面的文字，查看输出效果，但要注意，必须使用英文字符和标点，中文字符的处理我们后面再讲解。

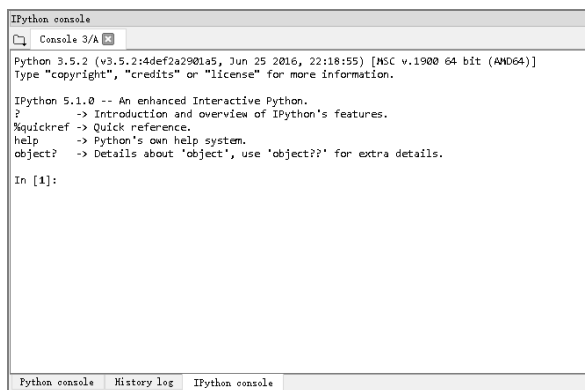


图 3-1 输出面板

3.1.1 简单调试

下面学习最简单的调试，如图 3-2 所示，我们去掉程序代码左边的引号，再单击“运行”按钮“▶”。

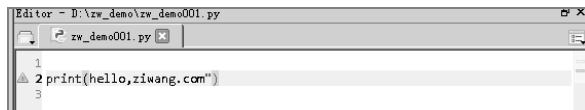


图 3-2 修改代码

此时，右下角的输出窗口显示如图 3-3 所示。

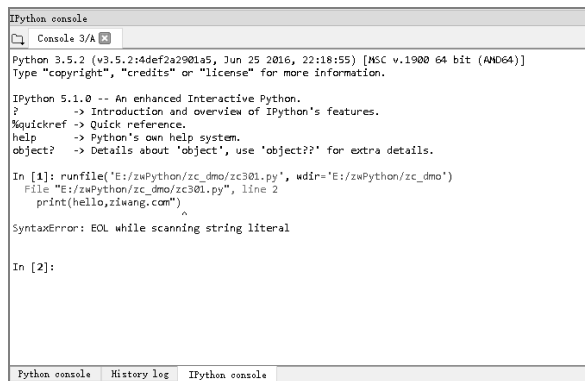


图 3-3 输出窗口

注意下面这行文字，表示有错误。

```
File "e:/zwPython/zc_demo/zc201.py", line 2
```

其中，“line 2”表示出错的代码位于第二行。

出错信息是：

```
SyntaxError: EOL while scanning string literal
```

以上代码表示字符串应用错误，我们加上引号即可。

3.1.2 控制台复位

有时，由于脚本代码或者其他原因，可能引发严重错误，系统运行时会出现死循环或崩溃。

如图 3-4 所示，单击 IDE 右侧的“Restart”下拉按钮，选择相应选项将控制台重新复位即可。

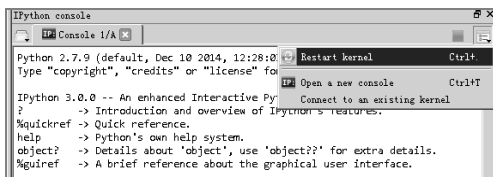


图 3-4 系统复位

3.2 案例3-2：增强版“hello,ziwang”

下面我们运行一个增强版的“hello,ziwang”。

(1) 单击工具栏中的“读取”按钮，打开“zc_demo\”目录下的 py302.py 脚本文件。

(2) 单击工具栏中部的绿色“运行”按钮“▶”。

案例 3-2 的脚本文件很简单，核心程序才十多行，不过功能非常强大，除输出文字信息外，还可提供中文输出检测系统多个重量级模块，如 OpenCV、Plotly、Pygame、Pandas 等，以及检测是否安装成功、版本是多少。

案例 3-2 代码如下：

```
# -*- coding: utf-8 -*-
```

```
import sys,os,re
import cv2
import arrow,plotly

import pandas as pd
import tushare as ts
import pygame

print("hello,zwPython 2017")
print("hello,TopQuant,TopFootball")
print("极宽量化回溯系统, 极宽足彩量化分析系统")
print("")
print("python ver:",sys.version)
print("")
print("re ver:",re.__version__)
print("arrow:",arrow.__version__)
print("plotly:",plotly.__version__)
print("")
print("pandas ver:",pd.__version__)
print("tushare ver:",ts.__version__)
print("")
print("pygame ver:",pygame.ver)
print("opencv ver:",cv2.__version__)
```

案例 3-2 如果运行无误，输出面板中的结果如下（不同版本的细节略有差别）：

```
hello,zwPython 2017
hello,TopQuant,TopFootball
极宽量化回溯系统, 极宽足彩量化分析系统

python ver: 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900
64 bit (AMD64)]

re ver: 2.2.1
arrow: 0.10.0
plotly: 2.0.0

pandas ver: 0.19.1
tushare ver: 0.6.2
```

```
pygame ver: 1.9.2b1
opencv ver: 3.1.0
```

案例 3-2 表明，即使是初学者，采用 BASIC 的过程模式编写简单代码，也能完成很复杂的功能。

3.3 案例3-3：列举系统模块库清单

案例 3-3 的文件名是 `zc_demo\py303mlst.py`，代码很简单，才十多行，代码如下：

```
# -*- coding: utf-8 -*-

import numpy as np
import scipy as sp
import pandas as pd
import pip

# =====
x10=pip.get_installed_distributions();
df=pd.DataFrame();
df['name']=x10
print(df.head())

df.to_csv('tmp/m10.csv',index=False)
```

案例 3-3 运行结果如下：

```
          name
0      3to2 1.1.1
1  zope.interface 4.3.2
2  zipline-cn-databundle 0.4
3          zict 0.1.0
4          zarr 2.1.3
```

运行后，会在 `tmp` 目录中生成一个名称为 `m10.csv` 的数据文件，文件里面就是当前 `zwPython` 所安装的第三方模块库的名称和版本号。《`zwPython` 用户手册》中 `zwPy3.5` 内置模块库列表就是这样完成的。

案例 3-3 的代码虽然简单，却调用了 `Pandas`（潘达思）数据分析模块，以及很少在程序中直接调用的 `pip` 模块库，来获取内置模块清单。

案例 3-3 是笔者在编程中实际使用的脚本程序，读者可以保存，查看自己的 Python 运行环境的模块库安装情况。

3.4 案例3-4：常用绘图风格

可视化计算是数据分析的重要组成部分，Python 可视化计算、数据分析、量化分析在传统上最重要的模块就是 Matplotlib。

如今，虽然新一代绘图模块 Plotly 横空出世，在互动性方面也具有压倒性的优势，但由于历史原因，目前的 Pandas（潘达思）数据分析模块和很多第三方内置绘图模块还是使用 Matplotlib 来绘制一般的简单图形，由此可见还是使用 Matplotlib 方便。

旧版本的 Matplotlib 默认的绘图风格与现代设计风格有些差距，因此需要增加一些扩展风格包。2017 年 1 月，期待已久的 Matplotlib 2.0 终于发布，在这些方面有所优化。

通常，Matplotlib 的绘图风格只有几种：

```
'bmh', 'dark_background', 'fivethirtyeight', 'ggplot', 'grayscale', 'default'。
```

zwPython 增加到二十余种：

```
'dark_background', 'seaborn-colorblind', 'classic', 'grayscale',  
'seaborn-dark-palette',  
'seaborn-ticks', 'fivethirtyeight', 'seaborn-paper', 'seaborn-poster',  
'seaborn-white',  
'seaborn-muted', 'seaborn-notebook', 'seaborn-dark', 'bmh', 'ggplot',  
'seaborn-darkgrid', 'seaborn-whitegrid', 'seaborn-bright',  
'seaborn-pastel',  
'seaborn-talk', 'seaborn-deep'
```

以上风格，除 Matplotlib 调用外，也可直接应用到 Pandas 绘图语句中，需要注意的是，不同的版本和不同的 Python 运行环境，以上具体参数在细节方面会有所差异。

案例 3-4 的文件名是 py304dr.py，代码使用 Matplotlib 绘图指令进行绘图，全部代码如下：

```
# -*- coding: utf-8 -*-  
  
import numpy as np  
import matplotlib as mpl  
import matplotlib.pyplot as plt
```



```
import pandas as pd

def dr_xtyp(_dat):
    #xtyp=['bmh','dark_background','fivethirtyeight','ggplot',
    'grayscale','default'];
    for xss in plt.style.available:
        plt.style.use(xss);print(xss)
        plt.plot(_dat['Open'])
        plt.plot(_dat['Close'])
        plt.plot(_dat['High'])
        plt.plot(_dat['Low'])
        fss="tmp\\stk001_"+xss+".png";plt.savefig(fss);
        plt.show()

# =====
df = pd.read_csv('dat\\apl12014.csv',index_col=0,parse_dates=[0],
encoding='gbk')
d30=df[:30];
dr_xtyp(d30);
```

运行后，会在 tmp 目录下生成一系列不同风格的图形，如图 3-5 所示，读者可以保存一下，作为日后编程的绘图参考。

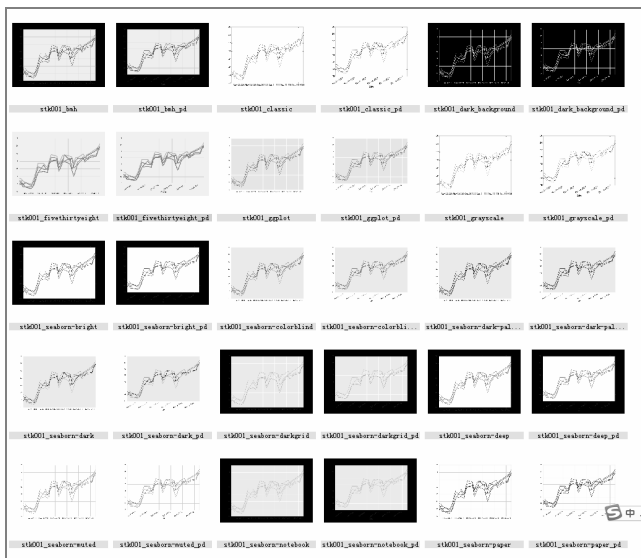


图 3-5 不同风格的输出图形

3.5 案例3-5：Pandas常用绘图风格

案例 3-5 的文件名是 py305drpd.py，使用 Pandas（潘达思）数据分析模块内置的 plot 命令绘制图形，代码如下：

```
# -*- coding: utf-8 -*-

import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd

def dr_xtyp(_dat):
    #xtyp=['bmh','dark_background','fivethirtyeight','ggplot',
'grayscale','default'];
    for xss in plt.style.available:
        plt.style.use(xss);print(xss)
        _dat['Open'].plot();
        _dat['Close'].plot();
        _dat['High'].plot();
        _dat['Low'].plot();
        fss="tmp\\stk001_"+xss+"_pd.png";plt.savefig(fss);
        plt.show()

# =====
df = pd.read_csv('dat\\apl2014.csv',index_col=0,parse_dates=[0],
encoding='gbk')

d30=df[:30];
dr_xtyp(d30);
```

案例 3-5 的运行结果与案例 3-4 类似，在此略过。

比较案例 3-4 和案例 3-5 两段程序，只有以下语句不同：

plt.plot(_dat['Open'])	_dat['Open'].plot();
plt.plot(_dat['Close'])	_dat['Close'].plot();
plt.plot(_dat['High'])	_dat['High'].plot();
plt.plot(_dat['Low'])	_dat['Low'].plot();

运行结果除 Pandas 优化的细节外,两者风格大体相同,具体细节请读者参看 tmp 目录下的相关图片。

Pandas (潘达思) 数据分析模块是最重要的数据分析工具, 必须了解其基本功能和常用的函数命令。

笔者特意增加了案例 3-5 来强调 Pandas (潘达思) 数据分析软件的使用, 如果读者对案例 3-5 有困惑, 请补习一下 Pandas (潘达思) 基础知识, 再继续深入学习。

3.6 案例3-6: 常用颜色表cors

下面介绍一些常用的颜色组合, 在极宽 zsys 模块中, 内置了多种常用的颜色组合, 以下是 zsys 模块颜色组设置的相关代码:

```
import matplotlib.colors
from matplotlib import cm

cors_brg=cm.brg(np.linspace(0,1,10))
cors_hot=cm.hot(np.linspace(0,1,10))
cors_hsv=cm.hsv(np.linspace(0,1,10))
cors_jet=cm.jet(np.linspace(0,1,10))
cors_prism=cm.prism(np.linspace(0,1,10))
cors_Dark2=cm.Dark2(np.linspace(0,1,10))
```

需要注意的是, 以上的颜色组代码没有使用常规的 rgb 参数设置模式, 而是使用 np.linspace 函数, 从 Matplotlib 的 Corlormap 中提取 10 个颜色, 组成一个系列的 10 种颜色, 如果用户的参数超过 10 种, 则系统会自动循环使用每组的 10 个颜色。

读者也可以参考以上 np.linspace 函数, 在本案例中, 使用 20、30、50 等其他参数, 看看具体效果有什么不同。

案例 3-6 的文件名是 py306_cors.py, 说明如何绘制 Matplotlib 内置的颜色图谱, 以及极宽量化模块内置的颜色组合。

颜色是图表设计的一个核心部分, 事实上, 现代图表与传统图表的一个重要差异就是颜色的组合, 最近 Matplotlib 2.0 升级版本, 其中最重要的升级之一, 就是颜色体系更加适合现代的设计风格。

案例 3-6 内置了以下两个函数:

- dr_cormap()函数, 绘制 Matplotlib 内置的颜色图谱, 不同的 Python 环境、Matplotlib

版本和辅助模块的差异, 具体数目会有所不同;

- `dr_cors_sys()`函数, 绘制极宽量化模块内置的颜色组合, 共 8 种。

`dr_cormap()`函数代码如下:

```
def dr_cormap(fcor='dat/cormap.dat'):
    clst=zt.f_lstRdTxt(fcor);
    ds=pd.Series(range(5,25));
    for xc,cor in enumerate(clst):
        css=cor[0]
        xss='cm.'+str(css)+'(np.linspace(0,1,10))'
        print(xc,'#',css,xss)
        cor2=eval(xss)
        #print(css,xss,cor2)
        ds.plot(kind='bar',rot=0,color=cor2)
        plt.savefig('tmp/cm_'+css+'.png')
```

运行后, 会在 `tmp` 目录下生成数十个不同的颜色组合图形文件, 读者可以选择自己喜欢的图形, 参考极宽的颜色设置代码, 添加自己喜欢的颜色组合。

案例 3-6 也是笔者工作中的实际代码程序, 读者可以浏览 `dat/cormap.dat` 文件, 查看相关的颜色参数数据, 保存好输出的图形文件, 特别是自己喜欢的一些颜色效果, 以供日后编程参考。

4

第 4 章

足彩量化分析系统

4.1 功能简介

极宽足彩量化分析系统，又称 Top Quant for Football，简称 TFB，是 Top 极宽量化开源团队根据 TopQuant 极宽量化系统，专门针对足彩行业，进行版本移植，可能是足彩领域首个专业的量化回溯系统。

极宽足彩量化分析系统属于全新的项目，没有任何可以参考的先例，唯一能够借鉴的，只有 TopQuant 量化分析软件。

期货、外汇、黄金、重金属等金融领域，只要修改一下数据源，基本上无需修改源码，就能直接使用 TopQuant 量化分析软件。而足彩的数据构成和博弈方式完全不同，需要采用全新的开发模式。

TFB 属于全功能、全免费开源软件，用户可在 TopQuant.vip 极宽量化社区“下载中心”免费下载软件源码、用户文档和配套的 tfbDat 足彩数据包。

TFB 是专业的足彩量化分析、回溯测试系统，内置人工智能、机器学习模块，提供策略分析和回溯测试功能，是一套全功能的、全开源的、纯 Python 量化策略分析系统，可以直接用于足彩实盘操作。

数据源方面，配合 tfbDat 足彩数据包和内置的足彩数据更新程序，用户可以完成足彩比赛数据和赔率数据的网络采集、数据清洗、自动去重和更新数据操作。

为了方便读者尽快熟悉 TFB 系统和配套的 tfbDat 足彩数据包,下面先介绍 TFB 系统和 tfbDat 数据包的基本结构。

极宽足彩量化分析系统正在不断开发和完善当中,和其他开源项目一样,未来版本收录的模块和源码也会不断发生变化,如果本书内容和具体源码有冲突,则以最新发布的软件源码为准。

4.1.1 目录结构

TFB 的文件目录结构如图 4-1 所示。

未来随版本变化,目录结构也可能会有变化,请读者注意。

其中,各个目录说明如下:

- demo, TFB 演示程序目录;
- source, TFB 程序源码。

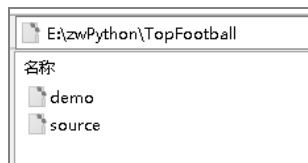


图 4-1 TFB 目录结构

4.1.2 TFB 安装与更新

TFB 极宽足彩量化分析系统的全部源码位于以下目录中 (x 是硬盘盘符):

```
x:\zwPython\TopFootball\source\
```

TFB 极宽足彩量化分析系统与 zwPython 进行了集成处理,可直接使用,支持 Python 3.x。

升级时,解压覆盖\zwPython\目录下的\TopFootball 子目录即可,注意,不要有双重 TopFootball 子目录。

移植时或使用其他 Python 环境时,可以把\TopFootball\source\目录下的脚本文件全部复制到自己的代码工作目录。移植时请注意代码中有关的数据文件目录设置。

如果采用 zcDat 极宽足彩数据包作为数据源,则 zcDat 与 zwPython 必须位于同一个硬盘根目录下。

强烈建议使用 SSD 固态硬盘,可提高 3~5 倍运行效率。

4.2 TFB主体框架

4.2.1 模块构成

在 TFB 极宽足彩量化分析系统 source 目录下，虽然文件繁多，但 TFB 系统构成非常简单，如图 4-2 所示。



图 4-2 TFB 系统构成

可以看出，TFB 主要由以下三大模块构成：

- Top-Base，极宽基础模块库；
- Top-Football，极宽足彩专业模块库；
- tfbDat，极宽足彩数据包。

4.2.2 Top-Base 极宽基础模块库

Top-Base 极宽基础模块库是极宽各种系统的基础模块库，各个模块文件均使用字母 z 作为文件首字母，一方面为了和各个应用模块区别，另一方面表示极宽系统源自最初的 zw 量化系统。

如图 4-3 所示是 Top-Base 极宽基础模块库的主要模块组成。



图 4-3 Top-Base 极宽基础模块库主要模块组成

Top-Base 极宽基础模块库包括以下模块。

- zsys，全局系统模块，定义全局参数、变量、常数等。
- ztools，常用工具函数库，缩写为 zt。
- ztools_str，常用字符串函数库，缩写为 zstr。
- ztools_web，常用 Web 网页抓取、htm 分析函数库，缩写为 zweb。
- zdraw，常用绘图函数库。
- ztop_data，极宽数据分析、预处理函数库，缩写为 zdat。
- ztop_ai，极宽通用机器学习模块库，缩写为 zai。

需要说明的是，未来随着极宽各个应用软件和系统的升级和发展，各个应用模块的构成可能会有所变动，请读者注意。

此外，因为字符串处理和 Web 网页数据抓取分析使用频率较多，所以从原来的 ztools 常用工具函数库中分离出来。

4.2.3 Top-Football 极宽足彩专业模块库

Top-Football 极宽足彩专业模块库是专门针对足彩行业开发的函数库，各个模块文件均使用字母 tfb 作为文件开头。如图 4-4 所示是 Top-Football 极宽足彩专业模块库的主要模块组成。



图 4-4 Top-Football 极宽足彩专业模块库主要模块组成

Top-Football 极宽足彩专业模块库包括以下模块。

- tfb_main，足彩主程序入口模块。
- tfb_sys，足彩系统模块库，包括足彩系统所需的类定义、全局参数和变量，缩写为 tfsys。
- tfb_tools，足彩系统常用工具函数库，缩写为 tft。

- tfb_draw, 足彩系统常用绘图函数库, 缩写为 tfdr。
- tfb_data, 足彩系统数据分析、预处理函数库, 缩写为 tfdat。
- tfb_strategy, 足彩系统常用策略模块库, 缩写为 tfsty。
- tfb_backtest, 足彩系统常用回溯分析函数库, 缩写为 tfbt。

为了方便读者使用与移植, 特意增加了一个 main 主模块, 定义了一个 main() 主函数入口, 类似 C 语言, 使程序入口一目了然。

需要说明的是, 未来随着极宽各个应用软件和系统的升级及发展, 各个应用模块的构成可能会有所变动, 请读者注意。

4.2.4 tfbDat 极宽足彩数据包

tfbDat 极宽足彩数据包原名 zcDat, 是国内首个足彩领域的开源大数据项目, 新版本的 tfbDat 极宽足彩数据包共收录了 2010—2017 年近七万场足球比赛的基本数据, 以及二百多万条赔率数据。

如图 4-5 和图 4-6 所示分别为 tfbDat 极宽足彩数据包的目录截图和文件截图。

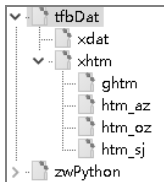


图 4-5 tfbDat 目录截图



图 4-6 tfbDat 文件截图

如图 4-5 所示的目录截图有以下几个特点。

- tfbDat、zwPython 默认位于同一个硬盘的根目录下。
- tfbDat 数据包共有两个子目录。
- xdat 子目录, 按比赛场次的 gid 收录各场比赛的赔率数据, 每场比赛一个文件。
- xhtml 是网页文件存档, 默认包括四个子目录。
- xhtml\ghtml\, 基本比赛数据网页文件, 按日期保存。
- xhtml\htm_oz\, 欧赔赔率网页文件, 按 gid 比赛场次保存, 每场比赛一个文件。
- xhtml\htm_az\, 亚赔赔率网页文件, 暂时未用, 参考欧赔资料。
- xhtml\htm_sj\, 数据分析网页文件, 暂时未用, 参考欧赔资料。

由图 4-6 所示的文件截图可以看到以下两个文件。

- gid2017.dat，比赛基本数据，每天自动更新网络数据。
- xdat2017.dat，赔率数据合集，是 xdat 目录下所有赔率数据的合集，主要用于系统建模，一般无需自己更新。

xdat2017 数据包共收录了两百多万条欧赔数据，数据量很大，更新时间很长，需要人工更新，Top 极宽量化开源团队会定期在极宽网站发布更新后的足彩数据包。

4.2.5 量化系统模块构成

Top-Football 极宽足彩专业模块库是专门针对足彩行业开发的函数库，也可以说是极宽量化系统的足彩移植版本，如图 4-7 所示是 Top-Quant 极宽量化系统模块构成。



图 4-7 Top-Quant 极宽量化系统模块构成

可以看出，Top-Quant 量化系统与 TFB 足彩系统结构基本相同，只是数据源和专业应用模块不同，底层的 Top-Base 极宽基础模块库更是完全相同。

如图 4-8 所示是 Top-Quant 极宽量化专业模块库的构成。



图 4-8 Top-Quant 极宽量化专业模块库构成

Top-Quant 极宽量化专业模块库包括以下模块。

- tq_main, 极宽量化分析主程序入口模块。
- tq_sys, 量化系统模块库, 包括系统所需的类定义、全局参数和变量, 缩写为 tqsys。
- tq_tools, 量化常用工具函数库, 缩写为 tqt。
- tq_draw, 量化常用绘图函数库, 缩写为 tqdr。
- tq_data, 量化数据分析、预处理函数库, 缩写为 tqdat。
- tq_strategy, 量化系统常用策略函数库, 缩写为 tqsty。
- tq_backtest, 量化常用回溯分析函数库, 缩写为 tqbt。

需要说明的是, 未来随着极宽各个应用软件和系统的升级及发展, 各个应用模块的构成可能会有所变动, 请读者注意。

为了方便读者使用与移植, 特意增加了一个 main 主模块, 定义了一个 main() 主函数入口, 类似 C 语言, 使程序入口一目了然。

4.2.6 案例 4-1: 赔率文件切割

xdat2017 数据包共收录了两百多万条欧赔数据, 数据量很大, 更新时间很长, 为了方便起见, 我们按年度时间将其切割为多个小文件。

案例 4-1 的文件名是 zc401_dat_cut.py, 介绍按指定的起始时间和终止时间切割数据包, 还介绍了使用 ztools 极宽工具函数包中的 timNSec 函数计算程序代码运行所消耗的时间。

案例 4-1 的全部代码请读者查看程序文件, 下面按程序里面的分组逐一进行介绍。

主流程第 1 组、第 2 组代码如下:

```
rs0='/tfbDat/'
fgid,fdat=rs0+'gid2017.dat',rs0+'xdat2017.dat'
tim0=arrow.now()
gids=pd.read_csv(fgid,index_col=False,dtype=str,encoding='gb18030')
tim2=arrow.now();t1=zt.timNSec(tim2,tim0);print('rd gid2017 #1,',t1)
xdatas=pd.read_csv(fdat,index_col=False,dtype=str,encoding='gb18030')
tim2=arrow.now();t1=zt.timNSec(tim2,tim0);print('rd xdat2017 #2,',t1)
#
```

以上代码主要是定义文件目录、文件名、初始时间，并分别读取 **gid** 数据文件和 **xdat** 赔率数据文件，对应的输出信息如下：

```
rd gid2017 #1, 0.24
rd xdat2017 #2, 14.15
```

	gid	cid		cname	pwin0	pdraw0	plost0	pwin9	pdraw9	plost9	vwin0	...	gtid	kwin	kwinrq	mplay	mtid	qj	qr	qs	tplay	tweek
	2290961	634246	818	Bwin.es	2.70	2.75	3.00	2.70	2.70	3.00	34.70	...	5	-1	-1	埃及	41	-1	0	-1	2017-02-05	0
	2290962	634246	820	Interwetten.es	2.55	2.70	2.90	2.70	2.65	2.80	35.41	...	5	-1	-1	埃及	41	-1	0	-1	2017-02-05	0
	2290963	634246	90005	gavg	2.65	2.78	2.95	2.75	2.72	2.92	35.09	...	5	-1	-1	埃及	41	-1	0	-1	2017-02-05	0
	2290964	634246	90009	gmax	3.10	3.05	3.15	3.00	3.00	3.20	38.00	...	5	-1	-1	埃及	41	-1	0	-1	2017-02-05	0
	2290965	634246	90001	gmin	2.20	2.30	2.45	2.20	2.30	2.45	30.79	...	5	-1	-1	埃及	41	-1	0	-1	2017-02-05	0

[5 rows x 35 columns]

案例 4-1 中的时间都是相对于程序运行开始的最初时间，每组代码运行的实际时间还需要减去前面所有代码的运行时间，后面几组输出信息也是如此。

由输出信息可以看出，**gid** 数据的读取速度很快，才 0.24 秒，读取 **xdat2017** 赔率数据文件的时间减去前面读取代码的时间，大约 13 秒。

xdat2017 数据文件有近 400MB，共收录了二百多万条赔率数据，可以参看输出信息。

案例 4-1 运行时，采用的是 64 位 Windows10 系统、Python 3.5 版本、E3-1230 的 CPU、主机内存 8GB，不过其中最大的功臣还是 SSD 固态硬盘。

程序代码第 3~6 组主要是切割数据文件，然后读取切割后的数据文件，代码如下：

```
print('')
tim0str,tim9str='2016-01-01','2016-12-31'
xd2016=fb_dat_cut(xdatas,tim0str,tim9str)
tim2=arrow.now();t1=zt.timNSec(tim2,tim0);print('cut xdat2016 #3,',t1)
#
xd2016.to_csv('tmp/xd2016.dat',index=False,encoding='gb18030')
tim2=arrow.now();t1=zt.timNSec(tim2,tim0);print('wr xdat2016 #4,',t1)
```

```
#
df2=pd.read_csv('tmp/xd2016.dat',index_col=False,encoding='gb18030')
tim2=arrow.now();t1=zt.timNSec(tim2,tim0);print('rd xdat2016 #5,',t1)
#
print('\nxdat2016.tail() #6')
print(df2.tail())
```

对应的输出信息是:

```
cut xdat2016 #3, 16.53
wr xdat2016 #4, 20.7
rd xdat2016 #5, 22.7

xdat2016.tail() #6
      gid   cid  cname pwin0 pdraw0 plost0 pwin9 pdraw9
plost9 vwin0 ...  gtid  kwin kwinrq mplay mtid  qj  qr  qs    tplay
tweek
    382537 628747  1119  博天堂.vu  10.00   6.00   1.17   7.0   5.75
1.22  8.92 ...   1046   0   -1   伊蒂哈  2027   2   0   3  2016-12-31   5
    382538 628747   103  Expekt  10.00   6.50   1.17   9.0   6.00
1.20  9.02 ...   1046   0   -1   伊蒂哈  2027   2   0   3  2016-12-31   5
    382539 628747  90005   gavg  10.70   6.36   1.17   9.0   6.36
1.20  8.70 ...   1046   0   -1   伊蒂哈  2027   2   0   3  2016-12-31   5
    382540 628747  90009   gmax  16.25   7.50   1.25  13.5   8.00
1.31 33.44 ...   1046   0   -1   伊蒂哈  2027   2   0   3  2016-12-31   5
    382541 628747  90001   gmin   1.01   1.02   1.01   6.7   2.00
1.10  5.74 ...   1046   0   -1   伊蒂哈  2027   2   0   3  2016-12-31   5

[5 rows x 35 columns]
```

xd2016.dat 是切割后的赔率数据文件,只有 2016 年度的数据,文件大小约 65MB,共收录了 38 万多条欧赔数据信息。

由以上输出信息可以看出:

- 切割完毕后程序的运行时间是 16.5 秒,减去前面代码运行的 14 秒,切割大约花了 2 秒;
- 保存 xd2016 数据文件花了 4 秒;
- 再次读取 xd2016 数据文件花了 2 秒,读取速度是写入速度的两倍。

最后一组代码为第 7 组,主要用于测试时间运算参数,代码如下:

```
print('')
tim2=arrow.now().shift(days=2)
t1,t2,t3=zt.timNSec(tim2,tim0),zt.timNHour(tim2,tim0),zt.timNDay(tim2,tim0)
print('s,h,d#9,',t1,t2,t3)
```

对应的输出信息是：

```
s,h,d#9, 172822.71 48.01 2.0
```

本书后面会介绍新一代的 Python 时间模块库 Arrow，此处使用的就是 Arrow 时间模块库，不过 Arrow 时间模块库缺少常用的时间间隔计算函数，所以我们在 ztools 常用工具函数库里面增加了一组函数，来计算相关的时间间隔。

- zt.timNSec(tim2,tim0)，按秒计算时间间隔。
- zt.timNHour(tim2,tim0)，按小时计算时间间隔。
- zt.timNDay(tim2,tim0)，按日计算时间间隔。

以上函数中的 tim2、tim0 参数格式为标准时间格式。

案例 4-1 中使用了自定义切割函数 fb_dat_cut，代码如下：

```
def fb_dat_cut(df,tim0str,tim9str):
    df2=df[tim0str<=df['tplay']]
    df3=df2[df2['tplay']<=tim9str]
    return df3
```

fb_dat_cut 函数根据数据列 tplay（比赛时间）切割数据，切割后的数据包含起始日期和结束日期。对日期进行比较时，采用的是字符串格式，而不是时间格式。

4.2.7 案例 4-2：批量切割数据文件

案例 4-1 使用的是自定义切割函数 fb_dat_cut，虽然运行无误，但是有些不够灵活，案例 4-2 使用修改后的通用切割函数 df_kcut8tim，使用更加方便灵活，可以根据用户指定的数据列进行切割。

案例 4-2 的文件名是 zc402_dat_cutx.py，核心代码如下：

```
import ztools_data as zdat
#-----

rs0='/tfbDat/'
fgid,fdat=rs0+'gid2017.dat',rs0+'xdat2017.dat'
```

```

gids=pd.read_csv(fgid,index_col=False,dtype=str,encoding='gb18030')
xdats=pd.read_csv(fdat,index_col=False,dtype=str,encoding='gb18030')
#
ksgn='tplay'
ylst=['2010','2011','2012','2013','2014','2015','2016']
#
ftg0='tmp/gidx_'
zdat.df_kcut8yearlst(gids,ksgn,ftg0,ylst)
#
ftg0='tmp/xd_'
zdat.df_kcut8yearlst(xdats,ksgn,ftg0,ylst)

```

运行后，会自动按指定的时间列表参数 ylst 输出 7 个 gid 比赛数据和 xdat 赔率数据，并输出相关的数据文件名：

```

tmp/gidx_2010.dat
tmp/gidx_2011.dat
tmp/gidx_2012.dat
tmp/gidx_2013.dat
tmp/gidx_2014.dat
tmp/gidx_2015.dat
tmp/gidx_2016.dat
tmp/xd_2010.dat
tmp/xd_2011.dat
tmp/xd_2012.dat
tmp/xd_2013.dat
tmp/xd_2014.dat
tmp/xd_2015.dat
tmp/xd_2016.dat

```

案例 4-2 的文件头导入了 ztools_data 极宽数据工具函数库：

```
import ztools_data as zdat
```

并调用其中的 df_kcut8yearlst 批量切割函数来分割数据文件：

```

def df_kcut8yearlst(df,ksgn,ftg0,yearlst):
    for ystr in yearlst:
        tim0str,tim9str=ystr+'-01-01',ystr+'-12-31'
        df2=df_kcut8tim(df,ksgn,tim0str,tim9str)
        ftg=ftg0+ystr+'.dat';print(ftg)
        df2.to_csv(ftg,index=False,encoding='gb18030')

```

`df_kcut8yearlst` 批量切割函数本身并不进行文件切割操作，而是设置好相关的参数，调用 `df_kcut8tim` 切割函数进行具体的切割工作：

```
def df_kcut8tim(df, ksgn, tim0str, tim9str):  
    df2=df[tim0str<=df[ksgn]]  
    df3=df2[df2[ksgn]<=tim9str]  
    return df3
```

`df_kcut8tim` 切割函数是案例 4-1 中自定义切割函数 `fb_dat_cut` 的升级版本，原来的切割函数代码如下：

```
def fb_dat_cut(df, tim0str, tim9str):  
    df2=df[tim0str<=df['tplay']]  
    df3=df2[df2['tplay']<=tim9str]  
    return df3
```

升级后的函数虽然只是增加了一个 `ksgn` 自定义数据列名称，但使用起来灵活很多。不仅可以根据不同的时间参数进行切割，甚至可以根据 `gid` 和比赛进行小范围切割，有兴趣的读者可以自己编程测试一下。

需要注意的是，`df_kcut8tim` 通用切割函数使用的时间和范围数据是字符串格式的，如果原数据是其他格式，调用前请用 Pandas 的 `astype(str)` 命令改为字符串格式。

案例 4-2 是笔者实际使用的工程代码程序，案例按年度切割数据，在后面的人工智能、机器学习环节当中都有实际使用，例如用 2015 年的赔率数据进行建模，再用 2016 年的数据进行测试验证，检查模型的模拟盘测试效果。

4.3 tfbDat数据结构

`tfbDat` 极宽足彩数据包主要包括两组数据：`gid` 比赛基本数据和 `xdat` 赔率数据。两组数据的汇总文件为：

- `gid2017.dat`，比赛基本数据，每天自动更新网络数据；
- `xdat2017.dat`，赔率数据合集，是 `xdat` 目录下所有赔率数据的合集，主要用于系统建模，一般无需自己更新。

`xdat2017` 数据包共收录了 200 多万条欧赔数据，有 400MB。在本书常规案例中一般无需全部加载，因此，我们在分析赔率数据时，通常使用前面介绍的年度分割赔率数据，一般是 `xd_2016.dat` 数据文件，在 `dat` 目录下面。

`gid2017.dat` 文件不大，才 5MB，一般不再使用分割数据包。

4.3.1 案例 4-3: tfb 数据格式

案例 4-3 的文件名是 `zc403_tfbdat.py`, 介绍 `gid` 基本比赛数据和 `xdat` 赔率数据的数据格式, 核心代码如下:

```
rs0='/tfbDat/'
fgid,fdat=rs0+'gid2017.dat','dat/xd_2016.dat'
#1
gids=pd.read_csv(fgid,index_col=False,dtype=str,encoding='gb18030')
print('gids')
print(gids.tail())
print(gids.columns)
#2
print('\nxdatas')
xdatas=pd.read_csv(fdat,index_col=False,dtype=str,encoding='gb18030')
print(xdatas.tail())
print(xdatas.columns)
```

案例 4-3 的核心代码分为两组, 分别输出 `gid` 基本比赛数据和 `xdat` 赔率数据的数据格式。

4.3.2 gid 基本比赛数据格式

第 1 组的输出数据如下:

```
gids
      gid gset mplay  mtid gplay  gtid  qj  qs qr kend kwin kwinrq tweek
tplay      tsell
68517  632918   智甲   巴勒人  2032  康塞普西  2823  -1  -1  0   0  -1
-1      0  2017-02-05  2017-02-06  00:55:00
68518  632919   智甲   天主大  1718   基约塔  6127  -1  -1  0   0  -1
-1      0  2017-02-05  2017-02-06  00:55:00
68519  633032  阿超杯    河床   864   拉努斯   963  -1  -1  0   0  -1
-1      6  2017-02-04  2017-02-05  00:55:00
68520  634245  非洲杯   布基纳    73    加纳    60  -1  -1  0   0  -1
-1      6  2017-02-04  2017-02-05  00:55:00
68521  634246  非洲杯    埃及    41   喀麦隆    5  -1  -1  0   0  -1  -1
```

```
0 2017-02-05 2017-02-06 00:55:00
```

```
Index(['gid', 'gset', 'mplay', 'mtid', 'gplay', 'gtid', 'qj', 'qs', 'qr',
'kend', 'kwin', 'kwinrq', 'tweek', 'tplay', 'tsell'], dtype='object')
```

第 1 组最后一行的数据就是对应 **gid** 基本比赛数据的数据格式：

```
Index(['gid', 'gset', 'mplay', 'mtid', 'gplay', 'gtid', 'qj', 'qs', 'qr',
'kend', 'kwin', 'kwinrq', 'tweek', 'tplay', 'tsell'], dtype='object')
```

在 `tfb_sys.py` 足彩系统参数模块中，对 **gid** 数据结构使用了预定义的字段名称和字段初始化数值。使用初始化数值的一个好处是，可以大体上规范各个字段的数据类型，无需再对每一个数据列进行类型定义，简化编程代码。

```
gidNil=['',' ',' ',' ',' ',' ',' ','-1','-1','0', '0','-1','-1', ' ',' ',' ']
gidSgn=['gid','gset','mplay','mtid','gplay','gtid', 'qj','qs','qr',
'kend','kwin','kwinrq', 'tweek','tplay','tsell']
```

其中，**gidSgn** 是 **gid** 比赛数据的字段名称，说明如下。

- **gid**，比赛场次 id 编码，是系统唯一的 id。
- **gset**，game-set 的缩写，联赛名称。
- **mplay**，main-play 的缩写，主队名称。
- **mtid**，main-team-id 的缩写，主队的编码 id。
- **gplay**，guest-play 的缩写，客队名称。
- **gtid**，guest-team-id 的缩写，客队的编码 id。
- **qj**，进球的拼音缩写，这个是国内行业惯例。
- **qs**，失球的拼音缩写，这个是国内行业惯例。
- **qr**，让球的拼音缩写，这个是国内行业惯例。本参数暂未使用，供日后扩展使用。
- **kend**，key for end 的缩写，1 代表比赛完结，有些取消的比赛一直用 0 代表。
- **kwin**，比赛胜负情况，3 为主队胜，1 为平局，0 为客队胜，默认和取消的比赛数值是-1。
- **kwinrq**，让球的比赛胜负情况，3 为主队胜，1 为平局，0 为客队胜，默认和取消的比赛数值是-1。本参数暂未使用，供日后扩展使用。
- **tweek**，time for week 的缩写，比赛星期换算，一般周末比赛最多。
- **tplay**，time for play 的缩写，比赛日期。
- **tsell**，time for sell 的缩写，彩票截止销售时间。

本书 **gid** 比赛基本数据文件的数据构成是参考 500 彩票网站的比赛数据网页(如图 4-9 所示)，同时结合实盘经验设计的。

<div>胜平负/让球</div>					<div>胜平负</div>	<div>让球胜平负</div>	<div>五大联赛</div>	<div>1.60赔率以下</div>	<div>更多选择</div>	<div>已截止(0场)</div>	<div>定制</div>	<div>赛事设置</div>	<div>2017-01-20</div>
编号	赛事	开赛时间	主队 VS 客队	让球	投注区			<div>竞彩奖金变化层</div>	数据	平均欧赔	排		
星期五 2017-01-20					31场比赛已截止								
					胜	平	负						
001	英超	16:50	[1] 悉尼FC 2:0 阿德莱 [9]	0	1.49	3.85	5.05	析亚欧	-	-			
				-1	2.54	3.50	2.22						
031	球会友谊	19:30	瓦勒伦 3:2 斯托曼	0	1.44	4.20	5.10	析亚欧	-	-			
				-1	2.30	3.70	2.36						
002	非洲杯	00:00	[2] 科特迪 2:2 民刚果 [1]	0	1.52	3.16	6.45	析亚欧	-	-			
				-1	3.10	2.85	2.20						
003	法乙	03:00	[15] 阿雅克 1:0 奥尔良 [20]	0	2.00	2.65	3.96	析亚欧	-	-			
				-1	4.60	3.40	1.62						
004	法乙	03:00	[9] 阿弗尔 1:2 阿雅GF [11]	0	2.10	2.60	3.75	析亚欧	-	-			
				-1	4.95	3.50	1.56						
005	法乙	03:00	[19] 欧塞尔 0:2 布尔格 [14]	0	2.07	2.70	3.62	析亚欧	-	-			
				-1	4.95	3.55	1.55						
006	法乙	03:00	[13] 尼奥特 2:1 亚眠 [4]	0	2.59	2.60	2.79	析亚欧	-	-			
				-1	6.45	4.00	1.38						
007	法乙	03:00	[6] 索肖 3:3 克莱蒙 [8]	0	1.88	2.65	4.55	析亚欧	-	-			
				-1	4.20	3.20	1.73						
008	法乙	03:00	[5] 斯特堡 图尔 [17]	0	1.65	3.05	5.15	析亚欧	-	-			
				-1	3.30	3.28	1.92						
009	法乙	03:00	[18] 拉瓦勒 1:0 特鲁瓦 [7]	0	2.52	2.62	2.85	析亚欧	-	-			
				-1	6.30	3.90	1.40						
010	法乙	03:00	[3] 兰斯 0:0 瓦朗谢 [10]	0	1.57	3.18	5.61	析亚欧	-	-			
				-1	3.34	2.96	2.03						

图 4-9 500 彩票网站的比赛数据网页

对比图 4-9 和 gid 数据列组成,可以发现,网页的部分信息如编号、球队排名等数据, gid 比赛基本数据文件都没有收录。

此外,在 500 彩票网站的比赛网页中,不仅提供了 gid 基本比赛数据,还提供了部分赔率数据。在某些彩票网站的比赛网页中,甚至提供了最常用的十大博彩公司,如必发、威廉、snai 等机构的赔率数据。这些彩票网站,只需抓取一个网页,就可获得 gid 比赛数据和 gdat 常用赔率数据,效率上快很多。

笔者采用的是多次提取模式,先抓取 gid 基本比赛数据,再根据 gid 比赛 id 编码,提取主流的赔率数据。

这样操作虽然效率低一点,但是在数据采集和未来扩展方面灵活很多,不仅可以提取更多的欧赔数据,而且可以提取亚赔、大小球及球队分析等各种数据。

4.3.3 xdat 赔率数据格式

第 2 组的输出数据如下:

```
xdatas
gid    cid    cname  pwin0 pdraw0 plost0  pwin9 pdraw9 plost9
```

```
vwin0 ... gtid kwin kwinrq mplay mtid qj qr qs tplay tweek
382537 628747 1119 博天堂.vu 10.00 6.00 1.17 7.00 5.75 1.22
8.92 ... 1046 0 -1 伊蒂哈 2027 2 0 3 2016-12-31 5
382538 628747 103 Expekt 10.00 6.50 1.17 9.00 6.00 1.20
9.02 ... 1046 0 -1 伊蒂哈 2027 2 0 3 2016-12-31 5
382539 628747 90005 gavg 10.70 6.36 1.17 9.00 6.36 1.20
8.70 ... 1046 0 -1 伊蒂哈 2027 2 0 3 2016-12-31 5
382540 628747 90009 gmax 16.25 7.50 1.25 13.50 8.00 1.31
33.44 ... 1046 0 -1 伊蒂哈 2027 2 0 3 2016-12-31 5
382541 628747 90001 gmin 1.01 1.02 1.01 6.70 2.00 1.10
5.74 ... 1046 0 -1 伊蒂哈 2027 2 0 3 2016-12-31 5
```

[5 rows x 35 columns]

```
Index(['gid', 'cid', 'cname', 'pwin0', 'pdraw0', 'plost0', 'pwin9',
'pdraw9', 'plost9', 'vwin0', 'vdraw0', 'vlost0', 'vwin9', 'vdraw9', 'vlost9',
'vback0', 'vback9', 'vwin0kali', 'vdraw0kali', 'vlost0kali', 'vwin9kali',
'vdraw9kali', 'vlost9kali', 'gplay', 'gset', 'gtid', 'kwin', 'kwinrq',
'mplay', 'mtid', 'qj', 'qr', 'qs', 'tplay', 'tweek'], dtype='object')
```

第 2 组最后一行的数据，就是对应 xdat 赔率数据的数据格式：

```
Index(['gid', 'cid', 'cname', 'pwin0', 'pdraw0', 'plost0', 'pwin9',
'pdraw9', 'plost9', 'vwin0', 'vdraw0', 'vlost0', 'vwin9', 'vdraw9', 'vlost9',
'vback0', 'vback9', 'vwin0kali', 'vdraw0kali', 'vlost0kali', 'vwin9kali',
'vdraw9kali', 'vlost9kali', 'gplay', 'gset', 'gtid', 'kwin', 'kwinrq',
'mplay', 'mtid', 'qj', 'qr', 'qs', 'tplay', 'tweek'], dtype='object')
```

在 tfb_sys.py 足彩系统参数模块中，对 xdat 数据结构使用了预定义的字段名称和字段初始化数值，相关代码如下：

```
gxdatNil=['', '', '', 0,0,0,0,0,0, 0,0,0,0,0,0, 0,0, 0,0,0,0,0,0,
'', '', '', '', '', '-1', '-1', '0', '-1', '-1', '', '' ]
gxdatSgn=['gid', 'cid', 'cname',
'pwin0', 'pdraw0', 'plost0', 'pwin9', 'pdraw9', 'plost9',
'vwin0', 'vdraw0', 'vlost0', 'vwin9', 'vdraw9', 'vlost9',
'vback0', 'vback9',
'vwin0kali', 'vdraw0kali', 'vlost0kali', 'vwin9kali', 'vdraw9kali',
'vlost9kali',
#
'gset', 'mplay', 'mtid', 'gplay', 'gtid',
'qj', 'qs', 'qr', 'kwin', 'kwinrq',
```

```
'tweek','tplay']
```

其中, `gxdatSgn` 是 `xdat` 赔率数据的字段名称, `gxdatSgn` 赔率数据对应的数据列有三十多列, 看起来有些复杂, 不过分组来看也很简单。

`gxdatSgn` 第 1 组数据包括 `gid`、博彩公司代码和名称:

```
gid,cid,cname,
```

其中:

- `gid`, 比赛场次 id 编码, 是系统比赛场次唯一的 id;
- `cid`, 博彩公司代码, 是系统博彩机构唯一的 id;
- `cname`, 博彩公司名称, 因为历史原因和翻译问题, 有些相同的 `cid`, 代表的博彩公司有多个不同的名称。

`gxdatSgn` 第 2 组数据包括开盘赔率和收盘赔率:

```
pwin0','pdraw0','plost0','pwin9','pdraw9','plost9,
```

- `pwin0`、`pdraw0`、`plost0` 分别是主队胜、平、负的开盘赔率。
- `pwin9`、`pdraw9`、`plost9` 分别是主队胜、平、负的收盘赔率。

`gxdatSgn` 第 3 组数据包括根据赔率计算的主队胜、平、负开盘概率和收盘概率:

```
vwin0','vdraw0','vlost0','vwin9','vdraw9','vlost9,
```

- `vwin0`、`vdraw0`、`vlost0` 分别是主队胜、平、负的开盘概率。
- `vwin9`、`vdraw9`、`vlost9` 分别是主队胜、平、负的收盘概率。

`gxdatSgn` 第 4 组数据是返回率, 即博彩奖金和总额的比例, 分为开盘返回率和收盘返回率:

```
vback0','vback9,
```

`gxdatSgn` 第 5 组数据是凯利指数:

```
vwin0kali','vdraw0kali','vlost0kali','vwin9kali','vdraw9kali','vlost9kali',
```

- `vwin0kali`、`vdraw0kali`、`vlost0kali` 分别是主队胜、平、负的开盘凯利指数。
- `vwin9kali`、`vdraw9kali`、`vlost9kali` 分别是主队胜、平、负的收盘凯利指数。

`gxdatSgn` 第 6 组、第 7 组、第 8 组数据都是 `gid` 基本比赛数据的复制。

`gxdatSgn` 第 6 组数据是比赛球队数据:

```
gset,mplay,mtid,gplay,gtid
```

- `gset`, `game-set` 的缩写, 联赛名称。
- `mplay`, `main-play` 的缩写, 主队名称。
- `mtid`, `main-team-id` 的缩写, 主队的编码 id。

- gplay, guest-play 的缩写, 客队名称。
- gtid, guest-team-id 的缩写, 客队的编码 id。

gxdatSgn 第 7 组数据是得分情况数据:

```
qj,qs,qr,kwin,kwinrq,
```

- qj, 进球的拼音缩写, 这个是国内行业惯例。
- qs, 失球的拼音缩写, 这个是国内行业惯例。
- qr, 让球的拼音缩写, 这个是国内行业惯例。本参数暂未使用, 供日后扩展使用。
- kwin, 比赛胜负情况, 3 为主队胜, 1 为平局, 0 为客队胜, 默认和取消的比赛数值是-1。
- kwinrq, 让球的比赛胜负情况, 3 为主队胜, 1 为平局, 0 为客队胜, 默认和取消的比赛数值是-1。本参数暂未使用, 供日后扩展使用。

gxdatSgn 第 8 组数据是比赛时间数据:

```
tweek,tplay
```

- tweek, time for week 的缩写, 比赛星期换算, 一般周末比赛最多。
- tplay, time for play 的缩写, 比赛日期。

本书 xdat 赔率数据格式的数据构成是参考 500 彩票网站的欧赔数据网页 (如图 4-10 所示), 同时结合实盘经验设计的。

数据分析 投注分析 百家欧赔 让球指数 亚盘对比 大小指数 比分指数 走势分析 技术统计														固定 单场
筛选	全部公司	即时水程			即时赔率			返还率			即时盈亏			统计
序号	赔率公司	定制	胜	平	负	胜	平	负	值	胜	平	负	胜	赔率下载
1	竞彩官方		1.66	3.65	3.85	52.72%	24.27%	23.01%	88.56%	0.88	0.92	0.87	主客同	
			1.62	3.70	4.15	54.70%	23.95%	21.35%	88.61%	0.85	0.93	0.94		
2	威廉希尔		1.85	3.75	4.00	51.13%	25.22%	23.65%	94.59%	0.97	0.95	0.90	主客同	
			1.85	3.75	4.00	51.13%	25.22%	23.65%	94.59%	0.97	0.95	0.90		
3	澳门		1.82	3.60	3.45	49.19%	24.87%	25.95%	89.52%	0.95	0.91	0.78	主客同	
			1.75	3.60	3.75	51.21%	24.89%	23.90%	89.62%	0.91	0.91	0.85		
4	立博		2.10	3.60	3.20	44.65%	26.05%	29.30%	93.77%	1.10	0.91	0.72	主客同	
			1.83	3.60	4.00	50.87%	25.86%	23.27%	93.05%	0.95	0.91	0.90		
5	Bet365		2.00	3.60	3.50	47.01%	26.12%	26.87%	94.03%	1.04	0.91	0.79	主客同	
			1.83	3.60	4.50	52.22%	26.54%	21.24%	95.56%	0.95	0.91	1.01		
6	Interwetten		1.75	3.65	3.75	51.38%	24.64%	23.98%	89.92%	0.91	0.92	0.85	主客同	
			1.75	3.60	3.90	51.68%	25.12%	23.19%	90.45%	0.91	0.91	0.88		
7	SNAI		1.95	3.50	3.45	47.12%	26.25%	26.63%	91.88%	1.02	0.88	0.78	主客同	
			1.75	3.65	4.00	52.17%	25.01%	22.82%	91.29%	0.91	0.92	0.90		
8	皇冠		1.91	3.70	3.25	47.53%	24.54%	27.93%	90.78%	1.00	0.93	0.73	主客同	
			1.81	3.70	4.35	52.49%	25.68%	21.84%	95.00%	0.94	0.93	0.98		
9	易胜博		1.83	3.50	3.70	49.57%	25.92%	24.52%	90.71%	0.95	0.88	0.83	主客同	
			1.82	3.70	3.60	50.06%	24.63%	25.31%	91.12%	0.95	0.93	0.81		
10	伟德		2.00	3.60	3.70	47.71%	26.50%	25.79%	95.42%	1.04	0.91	0.83	主客同	
			1.80	3.75	4.40	52.94%	25.41%	21.66%	95.28%	0.94	0.95	0.99		
11	Bwin		1.80	3.70	3.90	51.33%	24.97%	23.69%	92.40%	0.94	0.93	0.88	主客同	
			1.80	3.70	3.90	51.33%	24.97%	23.69%	92.40%	0.94	0.93	0.88		
12	Gamebookers		1.80	3.70	3.90	51.33%	24.97%	23.69%	92.40%	0.94	0.93	0.88	主客同	
			1.80	3.70	3.90	51.33%	24.97%	23.69%	92.40%	0.94	0.93	0.88		

图 4-10 欧赔数据网页

以前的 `zcDat` 数据包中，为节约空间，`xdat` 赔率数据没有收录 `gid` 比赛数据，导致每次需要 `gid` 中的数据时，都要重新调用 `gid` 数据包，无法直接使用 `Pandas` 的高速矢量运算函数，严重影响效率。

目前内存价格低廉，极宽网站推荐的量化服务器标配都是双 `E5-2670CPU` 和 `64GB` 内存的组合。而 `xdata` 赔率数据集成 `gid` 比赛数据后，整个数据文件包也不过 `400MB` 左右，一次性调入内存的时间也只要 `10` 秒左右。

这里也从侧面印证了笔者曾经的断言：足彩数据是最好的大数据测试对象。

4.4 足彩基本数据分析

有了 `gid` 比赛基本数据和 `xdat` 赔率数据包，我们就可以对足彩领域进行一些简单的基本分析。

4.4.1 案例 4-4：比赛数据基本图表分析

数据图表的核心是数据，有了数据，我们就能绘制相关的图表，进行进一步的分析。

`gid` 比赛基本数据虽然收录的信息不多，但也足以绘制一些简单而有趣的图表，同时，读者可以借助这些简单的数据，学习 `Python` 和 `Pandas`（潘达思）数据软件的最基本数据分析、汇总和统计功能。

案例 4-4 的文件名是 `zc404_gid_des.py`，介绍绘制简单的 `gid` 比赛数据图表，核心代码如下：

```
def gid_anz_top10(df, ksgn):

    xn9=len(df['gid'])
    d10=df[ksgn].value_counts()[:10];print(d10)

    #
    #---set chinese font
    mpl.rcParams['font.sans-serif'] = ['SimHei'] #指定默认字体
    mpl.rcParams['axes.unicode_minus'] = False #解决保存图像是负号 '-' 显示
    为方块的问题
```

```
#
d10.plot(kind = 'bar',rot=0,color=zensys.cors_brg)
plt.show()
#
dsum=d10.sum()
d10['other']=xn9-dsum
k10=np.round(d10/xn9*100,decimals=2)

k10.plot(kind = 'pie',rot=0,table=True)
plt.show()

#-----
rs0='/tfbDat/'
fgid=rs0+'gid2017.dat'
df=pd.read_csv(fgid,index_col=False,dtypes=str,encoding='gbk')
print(df.tail())
print('\n',df.describe())
#
gid_anz_top10(df,'gset')
```

案例 4-4 调用自定义的 `gid_anz_top10` 分析函数，分析 `gid` 比赛数据各个数据列排名前 10 位的数据，也就是常说的 Top10 数据分析。

案例 4-4 运行结果比较长，我们分段介绍，第 1 部分显示 `gid` 数据文件的尾部数据和调用 Pandas 的 `describe` 快速统计汇总函数，对 `gid` 数据文件做简单的统计：

	gid	gset	mplay	mtid	gplay	gtid	qj	qs	qr	kend	kwin	kwinrq	tweek
tplay			tsell										
	68517	632918	智甲	巴勒人	2032	康塞普西	2823	-1	-1	0	0	-1	
-1	0	2017-02-05	2017-02-06	00:55:00									
	68518	632919	智甲	天主大	1718	基约塔	6127	-1	-1	0	0	-1	
-1	0	2017-02-05	2017-02-06	00:55:00									
	68519	633032	阿超杯	河床	864	拉努斯	963	-1	-1	0	0	-1	
-1	6	2017-02-04	2017-02-05	00:55:00									
	68520	634245	非洲杯	布基纳	73	加纳	60	-1	-1	0	0	-1	
-1	6	2017-02-04	2017-02-05	00:55:00									
	68521	634246	非洲杯	埃及	41	喀麦隆	5	-1	-1	0	0	-1	-1
0	2017-02-05	2017-02-06	00:55:00										
	gid	gset	mplay	mtid	gplay	gtid	qj	qs	qr				


```

kend  kwin kwinrq tweek      tplay      tsell
      count    68522 68521 68521 68517 68521 68517 68522 68522 68522
68522 68522 68522 68522      68522      68522
      unique    68522   154  2007  1724  1958  1681   14   11   1
2      4      1      7      2530      14785
      top      438989   英甲   布里斯   653   布里斯   653   1   1   0
1      3      -1      6  2016-05-08  2013-02-08  23:30:00
      freq      1  3807   277   218   280   220  22219  23830  68522
68365 30810 68522 22940      103      131

```

新版本的 `describe` 函数对统计分析结果做了简化，默认没有分位数，只有简单的四行：`count`（数据总数）、`unique`（无重复总数）、`top`（最多数据）和 `freq`（最多的数据频率）。

由案例 4-4 的输出信息，我们通过对比赛数据进行最简单的分析，可以得出以下结论。

- 2010 年 1 月—2017 年 1 月，共有近 7 万场比赛。
- 在 `gset` 联赛数据中，共有 154 种不同的联赛，排名第一的是英甲，共有 3807 场比赛。
- 在球队中，按名称来看，主队 `mplay` 有 2007 支，客队有 1958 支，不过按球队 `id` 来看，主队 `mtid` 有 1724 支，客队 `gtid` 有 1681 支。这是因为翻译和历史原因，部分国外球队有几个不同的名称，但球队的 `gtid` 是唯一的。有趣的是，不管是主队还是客队，比赛次数最多的都是英甲的布里斯球队。
- `qj` 进球数据，有 14 种不同情况，其中最多的是进 1 球，有 22219 场。
- `qs` 失球数据，有 11 种不同情况，其中最多的是失 1 球，有 23830 场。
- `tweek` 数据列，显示星期六的比赛最多，有 22940 场。
- 按 `tplay` 比赛日期来看，2016-05-08 的比赛场次最多，当天有 103 场比赛。
- 按 `tsell` 博彩销售截止日期来看，2013-02-08 销售的比赛场次最多，当天有 131 场比赛。

案例 4-4 输出信息的第 2 部分是 Top10 的联赛信息数据：

```

英甲    3807
英冠    3779
日职乙  3033
阿甲    2744
巴甲    2642

```

英超	2598
西甲	2597
法甲	2595
意甲	2548
法乙	2378

英国不愧是足球强国，比赛场次最多的第一名英甲、第二名英冠都是英国的足球比赛，令人意外的是排名第三的是日本的足球联赛日职乙，比阿根廷、巴西的足球比赛场次还多。

案例 4-4 输出信息的第 3 部分是联赛数据 Top10 的柱形图，如图 4-11 所示。

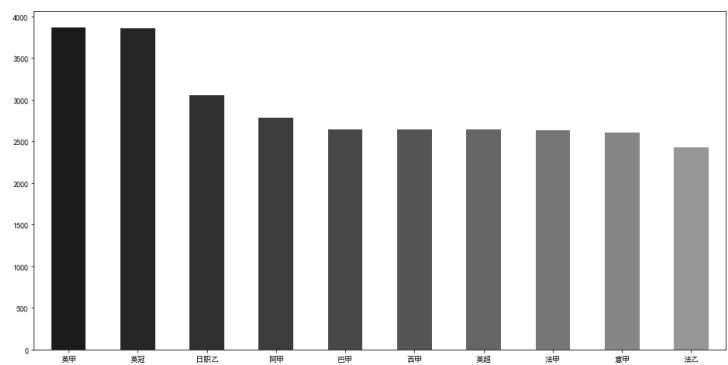


图 4-11 联赛数据 Top10 的柱形图

案例 4-4 输出信息的第 4 部分是联赛数据 Top10 的比例图，如图 4-12 所示。

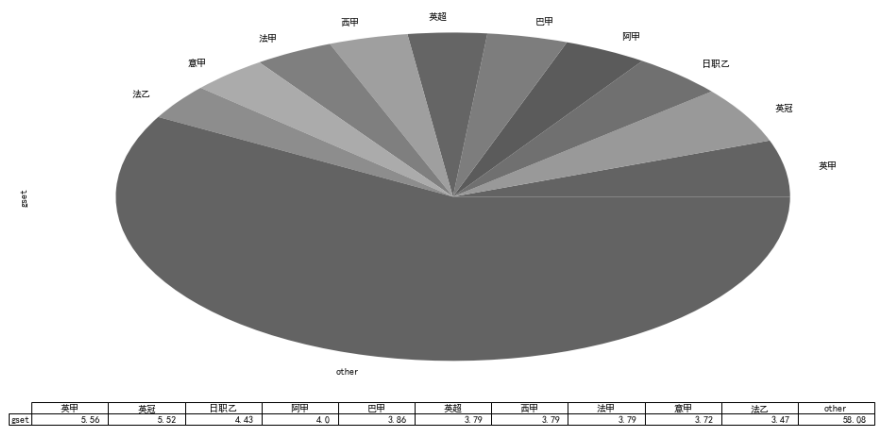


图 4-12 联赛数据 Top10 的比例图

为了进一步说明，第4部分的图形做了部分优化。

- 采用计算过的百分数据，表示相关的联赛比例。
- 英甲和英冠的比赛场次合计超过了比赛总场次的10%。
- 排名前10的联赛占据了42%的比赛场次，其他联赛的比赛场次合计约58%。

案例4-4在编程方面也使用了部分技巧。

使用Pandas内置的value_counts()函数统计排名靠前的联赛等各种数据：

```
d10=df[ksgn].value_counts()[:10];print(d10)
```

Pandas中也可以使用groupby分组函数，完成同样的工作：

```
gset9 =
df.groupby('gset').size().sort_values(ascending=False)[:25];print(gset9)
```

笔者喜欢采用简单的平面式编程架构，不喜欢层层叠叠的逻辑设计，以免引起混淆，所以一般不使用groupby分组功能，当然这只是笔者个人的编程习惯，仅供读者参考。

虽然目前的绘图模块主张使用新一代的Plotly互动绘图模块，不过Pandas内置的绘图模块还是基于Matplotlib的，一些简单的基本图形采用Pandas内置的plot绘图函数更加简单方便。

Matplotlib绘图函数的一个缺陷就是对中文的支持不好，默认无法正确显示中文，所以需要在程序中增加相关的中文字体设置语句：

```
#---set chinese font
mpl.rcParams['font.sans-serif'] = ['SimHei'] #指定默认字体
mpl.rcParams['axes.unicode_minus'] = False #解决保存图像是负号 '-' 显示为方块的问题
```

这段中文字体设置语句在调用Matplotlib模块前设置一次就可以了，后面的程序中无需再设置。

在案例4-4中，颜色设置使用的是以下脚本：

```
d10.plot(kind = 'bar',rot=0,color=zsyz.cors_brg)
```

这里面的具体技巧在本书案例3-5常用颜色表中已经介绍过了。

4.4.2 案例4-5：比赛数据进阶图表分析

前面的案例介绍了使用gid比赛数据来分析数据列gset联赛Top10数据，本节

我们参考案例 4-4 的模式，对 gid 数据文件中的其他数据列进行进一步的分析。

案例 4-5 的文件名是 `zc405_gid_anz.py`，核心代码如下：

```
rs0='/tfbdat/'
fgid=rs0+'gid2017.dat'
df=pd.read_csv(fgid,index_col=False,dtype=str,encoding='gbk')
tft.fb_df_type_xed(df)
df['qsum']=df['qj']+df['qs']
print(df.tail())
#
#tfd.dr_gid_top10(df,'gset')
tfd.dr_gid_top10(df,'mplay')
tfd.dr_gid_top10(df,'qj')
tfd.dr_gid_top10(df,'qs')
tfd.dr_gid_top10(df,'qsum')
tfd.dr_gid_top10(df,'tweek')
#-----
```

案例 4-5 与案例 4-4 的不同之处如下。

- 案例 4-4 的 `gid_anz_top10` 函数更名为 `dr_gid_top10`，保存在 `tfb_draw.py` 即 TOP 极宽量化绘图模块中。
- `qj` 进球、`qs` 失球等数据列改为 `int` 整数格式。
- 新增一个 `qsum` 总进球数据列，源自进球数和失球数的总和。

案例 4-5 运行结果很多，我们只对几个主要部分进行介绍。

首先，`mplay` 主队数据列 Top10 的数据如图 4-13 所示。

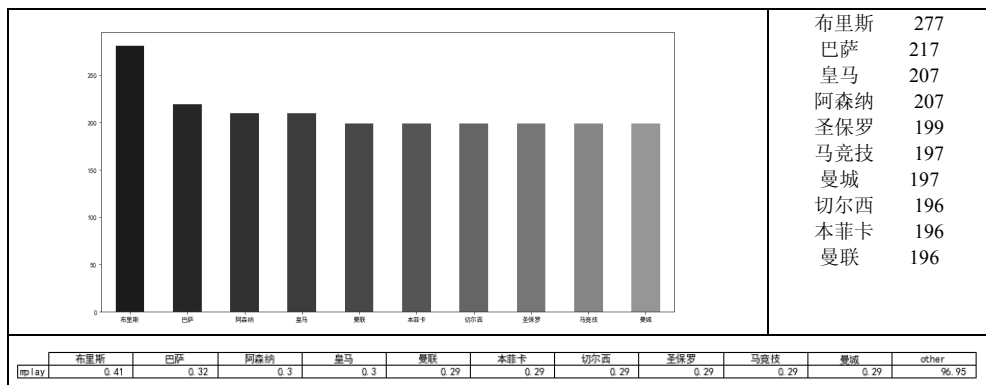


图 4-13 mplay 主队数据列 Top10 数据

新增的 qsum 总进球数数据图表如图 4-14 所示。

下面根据 `tweek` 数据列，看看星期的分布，如图 4-15 所示。

根据图 4-15，可以得出周六的比赛是最多的，占了 1/3，周日约占 25%，比赛最少的是周一，不到 5%。

案例 4-5 中调用了 tfb_tools 工具模块的 fb_df_type_xed 函数，把 qj 进球、qs 失球等数据列改为 int 整数格式，以便计算 qsum 总进球数的数据。

fb_df_type_xed 函数代码如下：

```
def fb_df_type_xed(df):
    df['qj']=df['qj'].astype(int)
    df['qs']=df['qs'].astype(int)
    df['qr']=df['qr'].astype(int)
    df['kwin']=df['kwin'].astype(int)
    df['kwinrq']=df['kwinrq'].astype(int)
```

4.4.3 案例 4-6：比赛数据年度图表分析

前面的案例都是历年 gid 比赛数据的综合分析。

案例 4-6 的文件名是 zc406_gid_anz2.py，介绍提取每年的数据，然后按年度分析有关的数据进展情况，主要代码如下：

```
def dr_gid_tim(df,ksgn,xlst):
    xdf=pd.DataFrame(columns=['nam','dnum'])
    ds=pd.Series(['',0],index=['nam','dnum'])
    for xtim in xlst:
        xtim0,xtim9=xtim+'-01-01',xtim+'-12-31'
        df2=df[xtim0<=df['tplay']]
        #print('\nx0',xtim,len(df2['gid']));#print(df2.tail())
        df3=df2[df2['tplay']<=xtim9]
        #print('x9',xtim,len(df3['gid']));#print(df3.tail())
        ds['nam'],ds['dnum']=xtim,len(df3['gid'])
        xdf=xdf.append(ds.T,ignore_index=True)
    #
    xdf.index=xdf['nam']
    print(xdf)
    xdf.plot(kind = 'bar',rot=0)
```

```
#-----
rs0='/tfbdat/'
fgid=rs0+'gid2017.dat'
df=pd.read_csv(fgid,index_col=False,dtype=str,encoding='gbk')
#
xlst=['2010','2011','2012','2013','2014','2015','2016']
dr_gid_tim(df,'gid',xlst)
```

案例 4-6 以 gid 比赛总场次为分析对象，按年度分析 2010—2016 年这几年来比赛场次数量的变化，具体情况如图 4-16 所示。

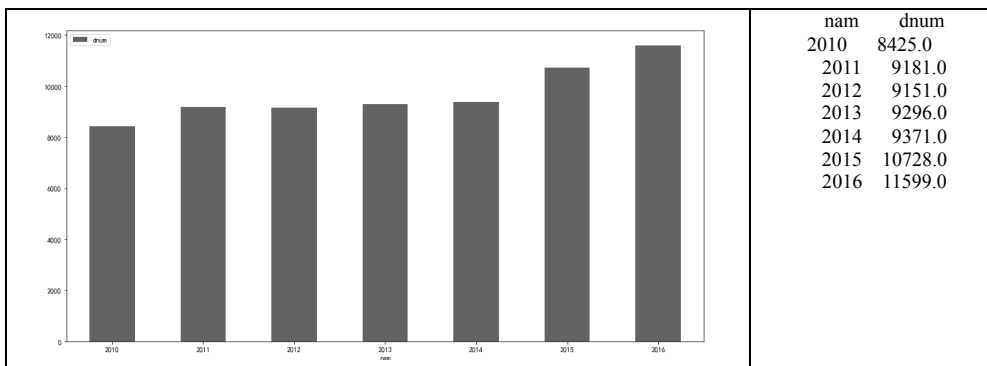


图 4-16 历年比赛场次分析图

由图 4-16 可以看出，从 2010 年到 2016 年，除了 2012 年比赛场次有略微减少以外，其余每年的比赛场次都是在稳定增长中，每年大约增长 5%~10%，从 2010 年的每年比赛 8400 多场次，到 2016 年每年比赛 1.15 万场次。

案例 4-6 只是以最基本的比赛场次介绍提取各年度数据，然后按年度分析相关的数据走势，其他更复杂的组合数据分析，如各个年度的进球和失球变化、球队变化，读者可以参考案例 4-6，自己编程尝试一下。

4.4.4 案例 4-7：比赛数据时间细分图表分析

前面的案例介绍的都是基本的 gid 比赛数据分析，案例 4-6 虽然进行了年度分析，但还是相对简单，下面介绍一些略微复杂的数据分析。

案例 4-7 的文件名是 zc407_gid_tim.py，介绍使用时间参数对 gid 数据进行细分。

案例 4-7 采用了多种不同的技巧，主流程代码分为 4 组，下面逐一进行介绍。

第 1 组代码如下：

```
rs0='/tfbdat/'
fgid=rs0+'gid2017.dat'
df=pd.read_csv(fgid,index_col=False,dtype=str,encoding='gbk')
#1
print('\n#1')
tim0=arrow.now()
dr_gid_tim_fx(df)
tn=zt.timNSec(arrow.now(),tim0)
print('#1,tn,',tn,',s')
```

第 1 组代码多了部分读取 gid 数据文件的代码，随后就是调用 dr_gid_tim_fx 自定义函数，按月提取 gid 比赛数据，并计算函数运行时间，相关运行结果如下：

```
#1
      nam    dnum
0   01  4802.0
1   02  4718.0
2   03  6603.0
3   04  7100.0
4   05  5480.0
5   06  2401.0
6   07  3206.0
7   08  7684.0
8   09  7546.0
9   10  7421.0
10  11  6623.0
11  12  4938.0
#1,tn, 64.23 ,s
```

第 1 组代码调用 dr_gid_tim_fx 自定义函数，其相关代码如下：

```
def dr_gid_tim_fx(df):
    xdf=pd.DataFrame(columns=['nam','dnum'])
    ds=pd.Series(['',0],index=['nam','dnum'])
    fx=lambda x:arrow.get(x).month
    for xc in range(1,13):
        xss='{0:02d}'.format(xc)
        df['month']=df['tplay'].apply(fx)
```



```

df2=df[df['month']==xc]
ds['nam'],ds['dnum']=xss,len(df2['gid'])
xdf=xdf.append(ds.T,ignore_index=True)
#
print(xdf)

```

在 `dr_gid_tim_fx` 自定义函数中，使用了几个小技巧：

```
fx=lambda x:arrow.get(x).month
```

和：

```
df['month']=df['tplay'].apply(fx)
```

结合 Pandas 的 `apply` 内置函数和 `lambda` 表达式，对 DataFrame 数据进行筛选。

`dr_gid_tim_fx` 是一种通用的 Pandas 数据筛选方法，如果遇到更加复杂的情况，可以把 `lambda` 表达式扩展为独立的函数。这种方法虽然可以用于各种复杂的场合，但在速度方面，因为不是纯矢量矩阵运算，所以无法利用 Pandas 内置的高性能优化函数，速度大约要慢 50 倍。

第 2 组代码调用的是 `dr_gid_tim` 自定义函数：

```

print('\n#2')
tim0=arrow.now()
dr_gid_tim(df)
tn=zt.timNSec(arrow.now(),tim0)
print('#2,tn,',tn,',s')

```

`dr_gid_tim` 自定义函数代码如下：

```

def dr_gid_tim(df):
    xdf=pd.DataFrame(columns=['nam','dnum'])
    ds=pd.Series(['',0],index=['nam','dnum'])
    for xc in range(1,13):
        xss,kss='{0:02d}'.format(xc),'-{0:02d}'.format(xc)
        df2=df[df['tplay'].str.find(kss)==4]
        ds['nam'],ds['dnum']=xss,len(df2['gid'])
        xdf=xdf.append(ds.T,ignore_index=True)
    #
    xdf.index=xdf['nam']
    print(xdf)
    xdf.plot(kind = 'bar',rot=0)
    plt.show()
    #

```

```
dsum=xdf['dnum'].sum()
xdf['k10']=np.round(xdf['dnum']/dsum*100,decimals=2)
xdf['k10'].plot(kind = 'pie',rot=0,table=True)
plt.show()
```

第 2 组代码的运行时间是：

```
#2,tn, 1.16 ,s
```

第 2 组代码的运行结果中不仅有文字信息，还有两幅图形，运行时间才 1 秒多，比第 1 组的运行速度快 50 倍，由此可见 Pandas 内置的矢量矩阵在速度方面的性能优势。

如图 4-17 和图 4-18 所示是第 2 组代码输出的图形。

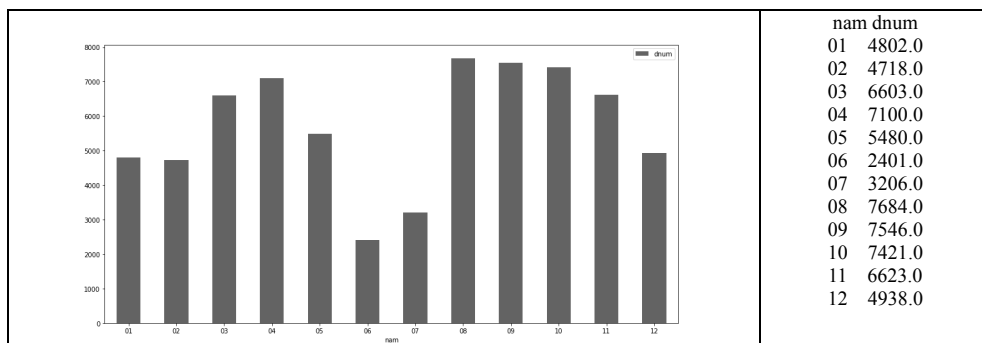


图 4-17 足彩比赛月度分布图

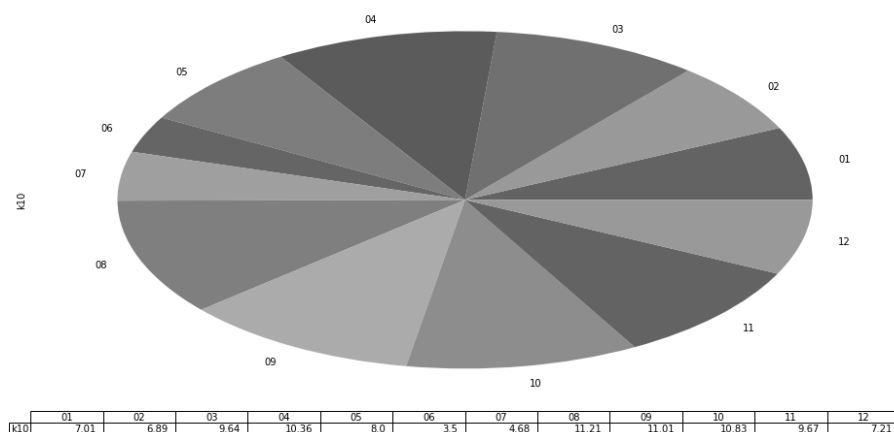


图 4-18 足彩比赛月度比例图

由图 4-17 和图 4-18 的足彩比赛月度分布图和比例图，我们可以清楚地看到，每年 8 月份的比赛最多，占 11.2%；每年 6 月份的比赛最少，才占年度比赛份额的 3.5%。

案例 4-7 的第 3 组代码如下：

```
#3
print('\n#3')
tim0=arrow.now()
ksgn='tplay'
xdf=zdat.df_get8tim(df,ksgn,'-',12,4)
zdr.dr_df_get8tim(xdf)
tn=zt.timNSec(arrow.now(),tim0)
print('#3,tn,',tn,',s')
```

第 3 组代码调用的是极宽模块库里面的函数 `zdat.df_get8tim` 和 `zdr.dr_df_get8tim`，这两个函数是第 2 组代码中 `dr_gid_tim` 自定义函数的优化版本，考虑到通用性，把数据筛选和绘图分开，作为两个函数。

`df_get8tim` 数据筛选函数位于 `ztools_data` 模块，代码如下：

```
def df_get8tim(df,ksgn,kpre,kn9,kpos):
#@ zdr.dr_df_get8tim
#
xdf=pd.DataFrame(columns=['nam','dnum'])
ds=pd.Series(['',0],index=['nam','dnum'])
for xc in range(1,kn9+1):
    xss,kss='{0:02d}'.format(xc),'{0}{1:02d}'.format(kpre,xc)
    df2=df[df[ksgn].str.find(kss)==kpos]
    ds['nam'],ds['dnum']=xss,len(df2['gid'])
    xdf=xdf.append(ds.T,ignore_index=True)
    #print(xc,'#',xss,kss)
#
xdf.index=xdf['nam']
return xdf
```

第 2 组的 `dr_gid_tim` 自定义函数的定义是：

```
def dr_gid_tim(df):
```

与第 2 组的 `dr_gid_tim` 自定义函数相比，`df_get8tim` 多了几个参数：

- `df`，数据源；
- `ksgn`，数据列名称；
- `kpre`，数据列查找前缀；

- kn9, 时间参数范围;
- kpos, 数据列查找, 正确匹配位置, 通常月份在字符串的位置是 4, 日期是 7。

dr_df_get8tim 绘图函数位于 zdraw 模块, 代码如下:

```
def dr_df_get8tim(xdf):
    #@zdat.df_get8tim
    #
    print(xdf)
    xdf.plot(kind = 'bar',rot=0)
    plt.show()
    #
    dsum=xdf['dnum'].sum()
    xdf['k10']=np.round(xdf['dnum']/dsum*100,decimals=2)
    xdf['k10'].plot(kind = 'pie',rot=0,table=True)
    plt.show()
```

因为 dr_df_get8tim 绘图函数与 df_get8tim 数据筛选函数之间关联度很大, 所以在两个函数代码中加入了调用注释。

第 3 组的运行结果和第 2 组的运行结果完全一样, 所以不再重复, 第 3 组代码的运行时间是:

```
#3,tn, 1.16 ,s
```

与第 2 组代码的运行时间完全一样, 可见在 Python 语言中, 函数在主模块还是子模块对于运行速度影响不大。

案例 4-7 的第 4 组代码如下:

```
#4
print('\n#4')
tim0=arrow.now()
xdf=zdat.df_get8tim(df,ksgn,'-',31,7)
zdr.dr_df_get8tim(xdf)
tn=zt.timNSec(arrow.now(),tim0)
print('#4,tn,',tn,',s')
```

第 4 组代码使用的是日期参数, 对 gid 比赛数据进行日期细分, 再绘制相关图表。第 4 组代码与第 3 组代码类以, 只是参数略微不同, 由此可见, 优化后的通用函数对于简化编程代码的好处。

第 4 组代码输出信息较长, 我们在此省略, 只提供两幅图表, 如图 4-19 和图 4-20 所示。

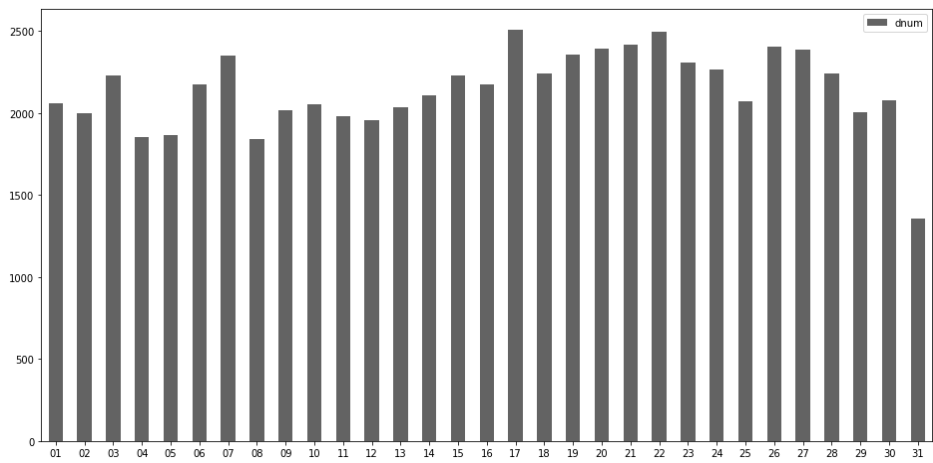


图 4-19 足彩比赛日期分布图

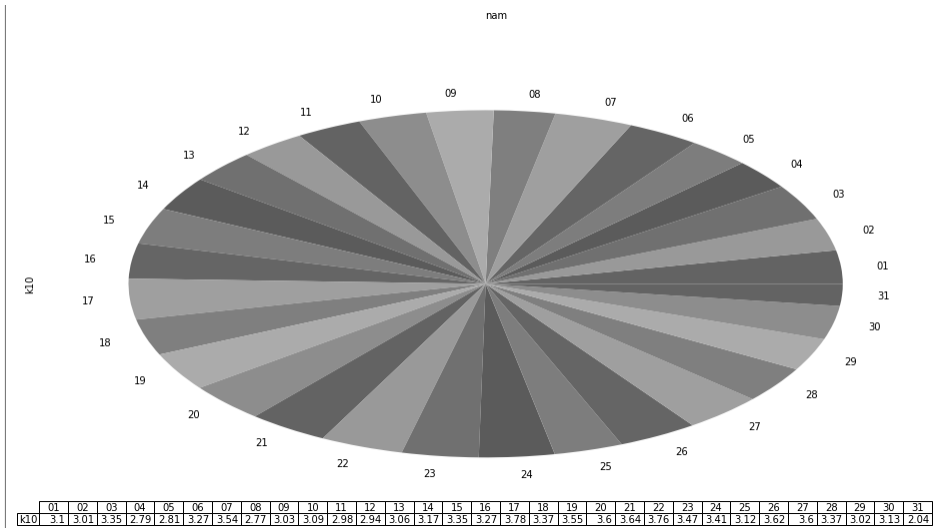


图 4-20 足彩比赛日期比例图

由图 4-19 和图 4-20 所示的足彩比赛日期分布图和比例图，我们可以看到，按日期分布的足彩比赛相对比较均匀，没有太大的差别，基本上每天的比赛都占 3%，其中比赛最少的是 31 日，才占 2%，这个可能与有的月份没有 31 日有关。

4.5 胜、平、负数据分析

4.5.1 案例 4-8：胜、平、负数据分析

案例 4-8 的文件名是 `zc408_pv01.py`，介绍根据 `gid` 比赛数据分析球队的胜、平、负情况。

案例 4-8 主流程代码分为两组，第 1 组代码如下：

```
#1
print('#1',)
#zdat.df_get8tim(df,ksgn,'-',12,4)
xdf=df_get8tim(df,ksgn,'-',12,4)
```

运行结果如下：

```
#1
tn,0.790s,fun:df_get8tim
```

第 1 组程序的输出数据是 `df_get8tim` 函数的运行时间。

4.5.2 @修饰符

程序中并没有使用 `ztools_data` 模块的 `df_get8tim` 数据切割函数，而是从模块库复制了一个主流程：

```
@ztst.fun_tim01
def df_get8tim(df,ksgn,kpre,kn9,kpos):
    #@ zdr.dr_df_get8tim
    #
    xdf=pd.DataFrame(columns=['nam','dnum'])
    ds=pd.Series(['',0],index=['nam','dnum'])
    for xc in range(1,kn9+1):
        xss,kss='{0:02d}'.format(xc),'{0}{1:02d}'.format(kpre,xc)
        df2=df[df[ksgn].str.find(kss)==kpos]
        ds['nam'],ds['dnum']=xss,len(df2['gid'])
        xdf=xdf.append(ds.T,ignore_index=True)
    #print(xc,'#',xss,kss)
```

```
#
xdf.index=xdf['nam']
return xdf
```

主流程的 `df_get8tim` 函数与模块库内的代码完全相同，只是在函数定义前，加了一个修饰符“@”，通过修饰符调用 `ztools_tst` 模块库的 `fun_tim01` 函数，计算自定义函数 `df_get8tim` 的运行时间。

因为 `@ztst.fun_tim01` 只能在函数定义前使用，所以把 `df_get8tim` 数据切割函数复制到主流程，作为演示。

`ztools_tst` 模块库的 `fun_tim01` 函数代码如下：

```
from functools import wraps

def fun_tim01(function):
    @wraps(function)
    def fun_tim(*args, **kwargs):
        t0 = arrow.now()
        result = function(*args, **kwargs)
        tn=zt.timNSec(arrow.now(),t0)
        print ('tn,{0:.3f}s,fun:{1}'.format(tn,function.__name__))
        return result
    return fun_tim
```

在 `fun_tim01` 函数定义中，也使用了@修饰符调用 `wrap` 函数。

@修饰符属于 Python 语言的魔法命令，可以简化程序设计，属于比较专业的范畴，如果读者不能理解，也没关系，一般编程无需使用，我们在此使用只是告知读者，在 Python 语言中有@修饰符这样一种语法命令，仅此而已。

此外，案例 4-8 文件头的 `import` 语句如下：

```
import ztools_tst as ztst
```

通过这条语句导入了 `ztools_tst` 模块库，如图 4-21 所示，细心的读者可能会发现，在 Top-Base 极宽基础模块库中，并没有这个 `ztools_tst` 模块库。



图 4-21 Top-Base 极宽基础模块库组成

这是因为 `ztools_tst` 模块库类似于 `zpd_talib` 模块库，都属于扩展模块库，并非软件架构中的标准模块，其中：

- `ztools_tst`，测试模块库，主要用于测试函数的运行性能，缩写为 `ztst`；
- `zpd_talib`，集成 Pandas 版本的 `talib` 进入函数库，缩写为 `ztalib`。

未来，随着项目的发展和升级，我们还会增加更多的扩展模块库。

4.5.3 胜、平、负分析

案例 4-8 的第 2 组代码用于计算主队的胜、平、负分布情况，代码如下：

```
print('#2',)
tfldr.dr_gid_top10(df, 'kwin')
```

第 2 组代码非常简单，类似前面的案例，使用 `kwin` 比赛结果数据列，作为参数直接调用 `dr_gid_top10` 函数，运行结果如图 4-22 和图 4-23 所示。

由图 4-22 和图 4-23 可以看出：

- 主队胜率最高，约占 45%；
- 两队踢平和主队负的比例差不多，在 23%~25%，这个结果出乎一般人的预料，大部分人认为足球很少踢平，胜或负的比例应该远远高于平局的比例；
- 还有 1%左右的比赛结果是 -1，表示流局或者还未开赛。

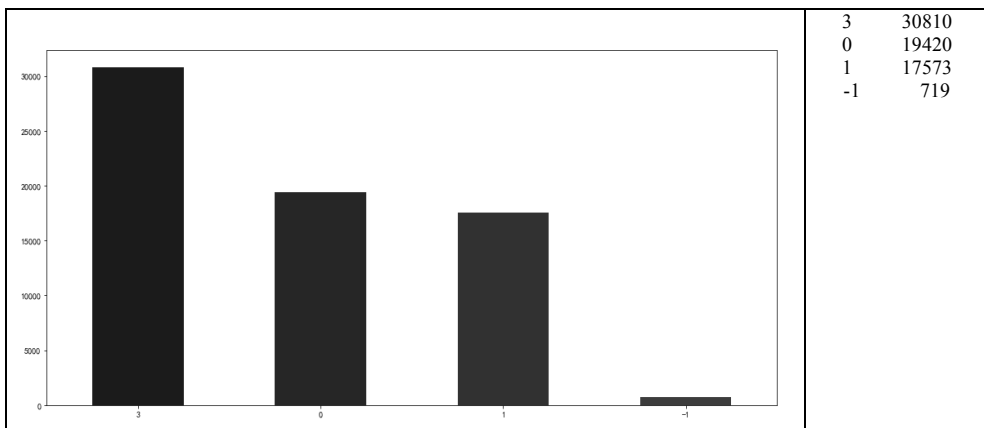


图 4-22 足彩比赛胜、平、负分布图

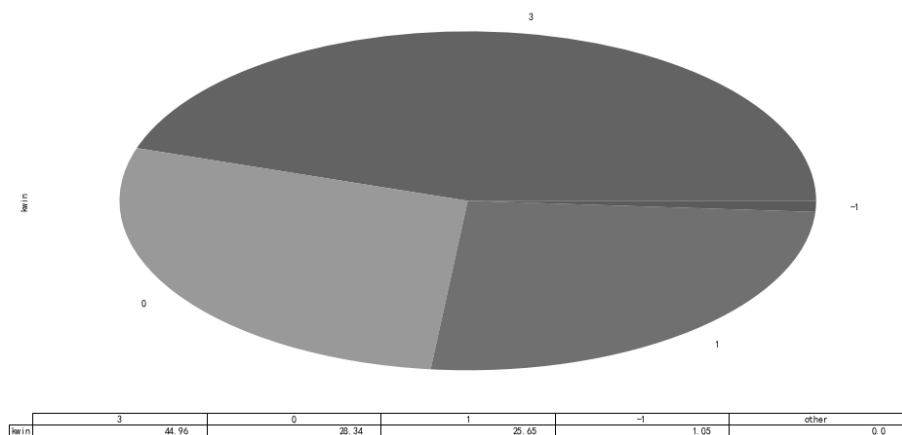


图 4-23 足彩比赛胜、平、负比例图

4.6 赔率数据分析

4.6.1 案例 4-9：赔率分析

案例 4-9 的文件名是 zc409_pv02.py，介绍赔率数据分析，核心代码如下：

```
rs0='/tfbdat/'
#fdat=rs0+'xdat2017.dat'
fdat='dat/xd_2016.dat'
df=pd.read_csv(fdat,index_col=False,dtype=str,encoding='gb18030')
dfk=df[df['cid']=='1']
#-----
xlst=['pwin0','pdraw0','plost0','pwin9','pdraw9','plost9',
      'vwin0','vdraw0','vlost0','vwin9','vdraw9','vlost9',
      'vback0','vback9',
      'vwin0kali','vdraw0kali','vlost0kali','vwin9kali','vdraw9kali',
      'vlost9kali']

for ksgn in xlst:
    print('\ndf',ksgn)
    tfdr.dr_gid_top10(df,ksgn,'tmp/'+ksgn+'_df_')
```

```
print('@dfk')
tfldr.dr_gid_top10(dfk,ksgn,'tmp/'+ksgn+'_dk_')
```

由以上代码可以看出，核心程序很简单，我们按流程逐一进行讲解。

首先将数据文件名读取到 `df` 变量，因为 `xdat2017` 历年的数据合集文件比较大，所以读者测试时，可以先用 2016 年度的数据文件 `dat/xd_2016.dat` 进行测试。

```
#fdat=rs0+'xdat2017.dat'
fdat='dat/xd_2016.dat'
df=pd.read_csv(fdat,index_col=False, dtype=str, encoding='gb18030')
```

然后提取官方赔率数据，`cid` 是我国国内官方赔率的机构代码。

```
dfk=df[df['cid']=='1']
```

`xlst` 是设置的测试参数，数据源自 `tfb_sys` 的全局变量定义，不过有所精简：

```
gxdatSgn=['gid','cid','cname',
          'pwin0','pdraw0','plost0','pwin9','pdraw9','plost9',
          'vwin0','vdraw0','vlost0','vwin9','vdraw9','vlost9',
          'vback0','vback9',
          'vwin0kali','vdraw0kali','vlost0kali','vwin9kali','vdraw9kali',
          'vlost9kali',
          #
          'gset','mplay','mtid','gplay','gtid',
          'qj','qs','qr','kwin','kwinrq',
          'tweek','tplay']
```

最后是主代码：

```
for ksgn in xlst:
    print('\ndf',ksgn)
    tfldr.dr_gid_top10(df,ksgn,'tmp/'+ksgn+'_df_')
    print('@dfk')
    tfldr.dr_gid_top10(dfk,ksgn,'tmp/'+ksgn+'_dk_')
```

主代码根据 `xlst` 循环设置 `ksgn`，调用 `zdraw` 模块的 `dr_gid_top10` 函数，方便绘制相关的图形。

4.6.2 扩充 `dr_gid_top10` 绘图函数

需要补充说明的是，`dr_gid_top10` 绘图函数原来是没有图片保存功能的，在此，笔者对函数进行了扩充，扩充后的代码如下：

```
def dr_gid_top10(df,ksgn,ftg0=''):

    xn9=len(df['gid'])
    d10=df[ksgn].value_counts()[:10];print(d10)

    #
    #---set chinese font
    mpl.rcParams['font.sans-serif'] = ['SimHei'] #指定默认字体
    mpl.rcParams['axes.unicode_minus'] = False #解决保存图像是负号 '-' 显示
为方块的问题
    #
    d10.plot(kind = 'bar',rot=0,color=zensys.cors_brg)
    if ftg0!='':plt.savefig(ftg0+'_bar.png')
    plt.show()
    #
    dsum=d10.sum()
    d10['other']=xn9-dsum
    k10=np.round(d10/xn9*100,decimals=2)

    k10.plot(kind = 'pie',rot=0,table=True)
    if ftg0!='':plt.savefig(ftg0+'_pie.png')
    plt.show()
```

扩充 `dr_gid_top10` 绘图函数主要修改以下地方。

函数定义增加了一个 `ftg0` 参数:

```
def dr_gid_top10(df,ksgn,ftg0=''):
```

增加了两条图片文件保存语句:

```
if ftg0!='':plt.savefig(ftg0+'_bar.png')
```

和:

```
if ftg0!='':plt.savefig(ftg0+'_pie.png')
```

4.6.3 赔率对比

下面根据案例 4-9 的输出图片, 查看部分运行结果, 如表 4-1 所示。

需要注意的是:

- 表 4-1 中的 `df`、`dfk` 分别是案例 4-9 中的变量名称, 其中 `df` 是总数据, `dfk` 是官方赔率数据;

- ● ●

表 4-1 赔率运行结果

名称	df 开盘	dfk 开盘	df 收盘	dfk 收盘	说明
pwin	21.4%	9.8%	19.7%	10.5%	主队胜赔率
pdraw	55%	50.5%	48.7%	56.4%	主队平赔率
plost	17.3%	12.7%	16.6%	11.1%	主队负赔率
vwin-kalii	58.4%	73.4%	68.8%	79%	主队胜凯利指数
vdraw-kali	67.3%	80.5%	69.5%	80%	主队平凯利指数
vlost-kali	45.2%	57.2%	55.7%	62.1%	主队负凯利指数

案例 4-9 部分运行结果如图 4-24~图 4-27 所示。

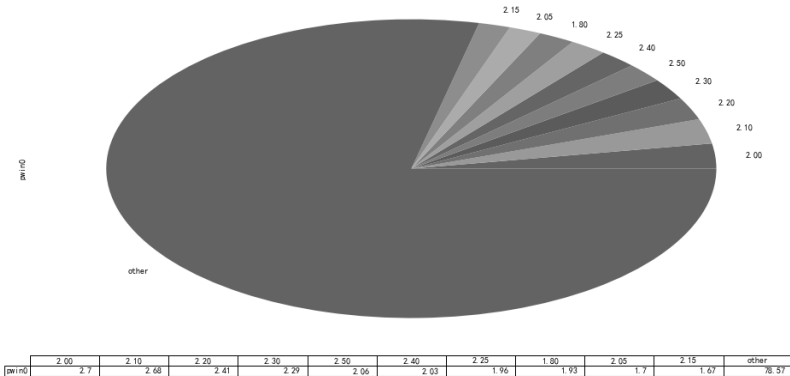


图 4-24 df 主队胜开盘赔率

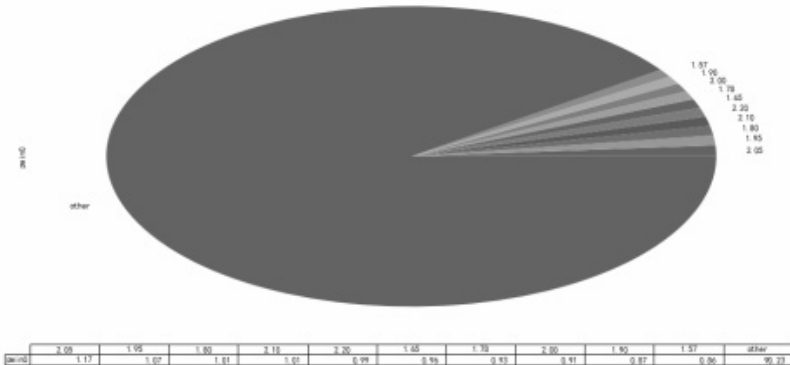


图 4-25 dfk 主队胜开盘赔率

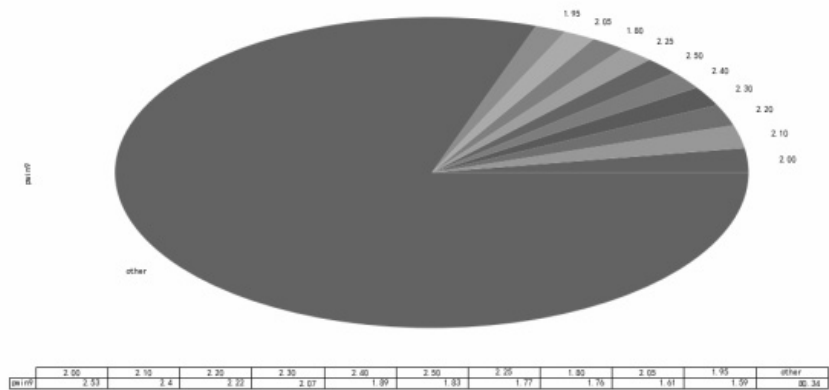


图 4-26 df 主队胜收盘赔率

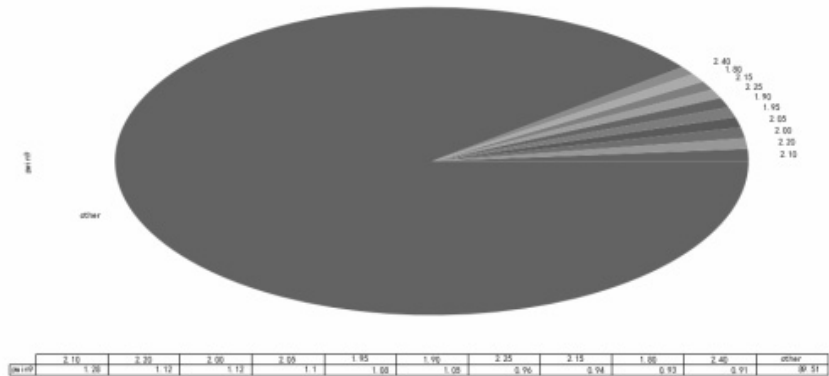


图 4-27 dfk 主队胜收盘赔率

5

第 5 章

常用数据分析工具

足彩分析，无论是赔率分析还是胜、平、负分析，归根结底都是数据分析，本章和后面几章将介绍 Python 数据分析的各种知识，包括数据采集、网页抓取、数据清洗整理、数据可视化，以及目前大数据领域流行的人工智能、机器学习等。

同时，还会介绍常用的辅助工具库，以提高数据分析的工作效率，当然这里介绍的各种模块库和工具软件都是基于 Python 编程语言的。

5.1 Pandas 数据分析软件

5.1.1 Pandas 简介

Python 数据分析最核心的就是 Pandas（潘达思）数据分析软件，所以笔者将其单独作为一节，并放在第一节进行介绍。

如图 5-1 所示是 Pandas 网站首页截图，网址是 <http://pandas.pydata.org/>。

Pandas 数据分析软件，全名是 Python Data Analysis Library，内置大量数据分析函数库和一些标准的数据模型，提供高效操作大型数据集所需的工具。

Pandas 的名称来自于数据面板(panel data)和数据分析(data analysis)。panel data 是经济学中关于多维数据集的一个术语，在 Pandas 中也提供了 panel 的数据类型。

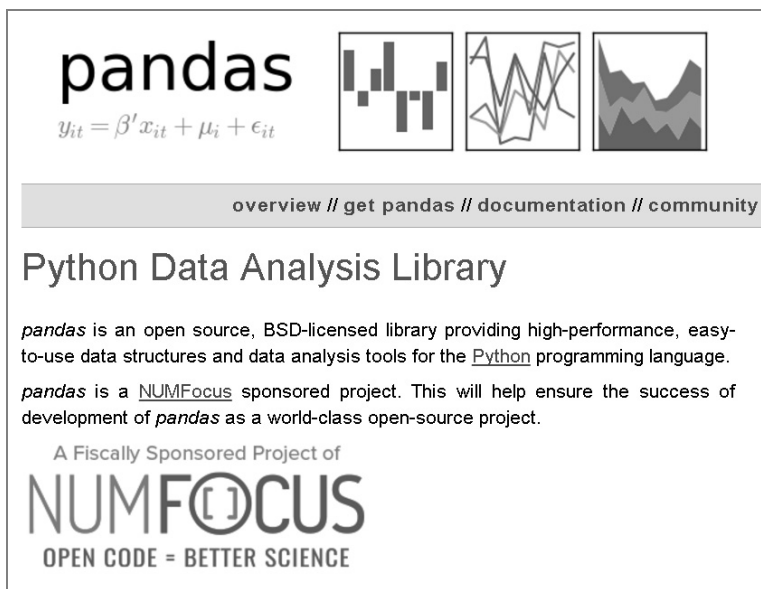


图 5-1 Pandas 网站首页截图

Pandas 提供大量能使我们快速、便捷地处理数据的函数和方法，这是使 Python 成为强大而高效的数据分析环境的重要因素之一。

Pandas 是 Python 的一个数据分析包，最初由 AQR Capital Management 于 2008 年 4 月开发，并于 2009 年年底开源，目前由专注于 Python 数据包开发的 PyData 开发团队继续开发和维护，属于 PyData 项目的一部分。

Pandas 最初被作为金融数据分析工具而开发，因此，Pandas 为时间序列分析提供了很好的支持。

Pandas 是基于 NumPy 和 Matplotlib 开发的，主要用于数据分析和数据可视化，它的数据结构 DataFrame 和 R 语言里的 data.frame 很像，特别是对于时间序列数据有自己的一套分析机制，非常方便高效。

pydata.org 是目前数据分析行业最重要的网站之一，特别是对于 Python 数据分析，其产品下载页面如图 5-2 所示。目前业内最重要的数据分析软件包基本上都是由 pydata 网站和团队在维护，如 Pandas、Numpy、Scipy、SymPy、Stasmodles、Numba、iPython、Matplotlib 和 Scikit-learn 等。

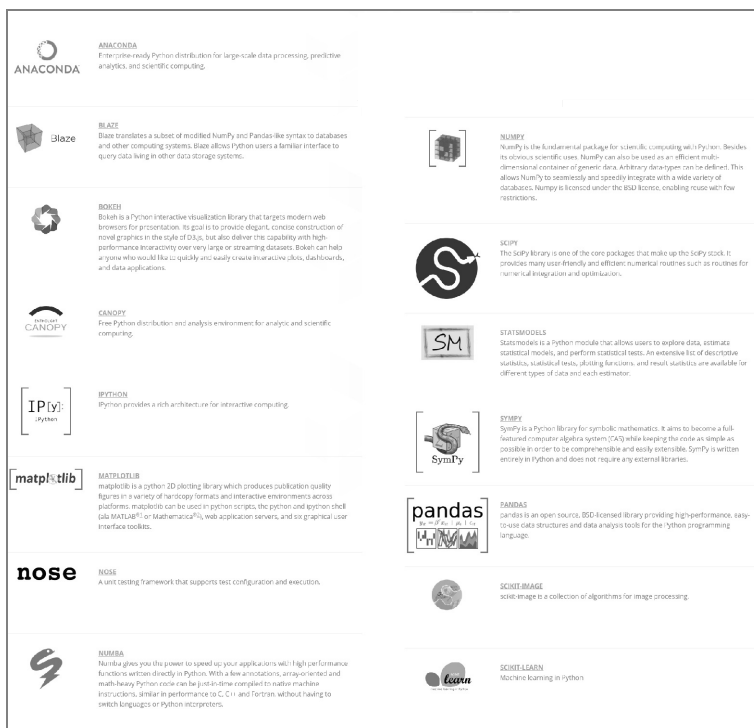


图 5-2 pydata.org 产品下载页面

Pandas 数据分析软件内置的数据结构有以下几个。

- **Series**: 一维数组，与 Numpy 中的一维 Array 类似，二者与 Python 基本的数据结构 List 也很相近，其区别是 List 中的元素可以是不同的数据类型，而 Array 和 Series 中只允许存储相同的数据类型，这样可以更有效地使用内存，提高运算效率。
- **Time-Series**: 以时间为索引的 Series。
- **DataFrame**: 二维的表格型数据结构。很多功能与 R 语言中的 data.frame 类似，可以将 DataFrame 理解为 Series 的容器。
- **Panel**: 数据面板，三维的数组，可以理解为 DataFrame 的容器。

虽然 Pandas 数据分析软件功能十分强大，但数据分析只靠 Pandas 软件，对于复杂多变的数据分析明显还是不够的，必须配合其他多种模块库，共同完成各种具体实盘工作。

Python 数据分析常用的模块库包括：科学计算、人工智能、数据清洗、统计分

析、金融数据采集、语义分析和数据可视化等。

5.1.2 案例 5-1: Pandas 常用统计功能

我们分析足彩的赔率数据和胜、平、负的目的是为了从这些数据、信息当中寻找各种相关的特征模式，建立量化分析模型。

本节介绍几种最常用的衍生数据，如最大值、最小值、中位数等。

首先介绍 Pandas 内置的常用统计函数。

- `count`，计算非 NA 值的数量。
- `describe`，针对 Series 或 DF 列计算汇总统计。
- `min`、`max`，最小值和最大值。
- `argmin`、`argmax`，最小值和最大值的索引位置（整数）。
- `idxmin`、`idxmax`，最小值和最大值的索引值。
- `quantile`，样本分位数（0 到 1）。
- `sum`，求和。
- `mean`，均值。
- `median`，中位数。
- `mad`，根据均值计算平均绝对离差。
- `var`，方差。
- `std`，标准差。
- `skew`，样本值的偏度（三阶矩）。
- `kurt`，样本值的峰度（四阶矩）。
- `cumsum`，样本值的累计和。
- `cummin`、`cummax`，样本值的累计最小值和累计最大值。
- `cumprod`，样本值的累计积。
- `diff`，计算一阶差分（对时间序列很有用）。
- `pct_change`，计算百分数变化。

学习以上部分函数需要有一定的统计学基础，但它们都已经被包装为普通的函数，在此我们不做深入探讨，读者知道如何使用就可以了，对于函数背后的理论，有兴趣的读者可以自行研究。

案例 5-1 的文件名是 `zc501_pdx01.py`，通过具体的代码介绍使用以上函数分析足彩赔率数据。

案例 5-1 代码较长，我们还是分组逐一讲解。

案例 5-1 的第 1 组和第 2 组代码如下：

```
rs0='/tfbdat/'
#fdat=rs0+'xdat2017.dat'
fdat='dat/xd_2016.dat'
df=pd.read_csv(fdat,index_col=False,dtype=str,encoding='gb18030')
gid='512751'
dfk=df[df['gid']==gid]
df2=dfk[dfk['cid']<'90000']

print('\n#1')
print(df2.describe())

print('\n#2')
print(df2.tail())
```

以上代码首先是设置数据文件名，然后读入系统，根据制定的 `gid`，筛选 `gid` 匹配的数据到 `dfk` 变量，再根据博彩机构代码，过滤 90000 以上的几个统计数据，即平均值、最大值和最小值：

```
df2=dfk[dfk['cid']<='90000']
```

然后调用 Pandas 的小计函数 `describe`，输出 `df2` 的小计结果和尾部数据：

```
#1
      gid  cid  cname pwin0 pdraw0 plost0 pwin9 pdraw9 plost9
vwin0 ... gtid kwin kwinrq mplay mtid  qj  qr  qs      tplay tweek
count      30   30      30   30      30   30   30   30      30   30
30 ...    30   30      30   30   30  30  30  30      30   30
unique      1   30      30   19   12   12   17   13   17
28 ...    1    1      1    1    1  1  1  1      1    1
top      512751 140 Bet-52 2.95 3.30 2.20 4.10 3.40 1.95
29.28 ...   905    0    -1 黑山 21 4928 0 0 3 2016-09-02    4
freq      30    1      1    4    7    8    3    7    4    2 ...
30  30      30   30      30  30  30  30      30   30

[4 rows x 35 columns]
```

```
#2
      gid  cid  cname pwin0 pdraw0 plost0 pwin9 pdraw9 plost9
vwin0 ... gtid kwin kwinrq mplay mtid qj qr qs      tplay tweek
      25 512751 66 Betshop 2.95 3.40 2.20 4.20 3.45 1.80
31.17 ... 905 0 -1 黑山 21 4928 0 0 3 2016-09-02 4
      26 512751 67 Betsson 2.95 3.35 2.15 3.85 3.25 1.87
30.74 ... 905 0 -1 黑山 21 4928 0 0 3 2016-09-02 4
      27 512751 1001 Leon 3.92 3.57 1.90 4.66 3.72 1.88
24.03 ... 905 0 -1 黑山 21 4928 0 0 3 2016-09-02 4
      28 512751 734 Smarkets 2.22 2.48 1.76 4.20 3.65 2.02
31.68 ... 905 0 -1 黑山 21 4928 0 0 3 2016-09-02 4
      29 512751 1304 Bet-52 2.92 3.31 2.20 3.55 3.38 1.92
31.16 ... 905 0 -1 黑山 21 4928 0 0 3 2016-09-02 4

[5 rows x 35 columns]
```

第3组代码输出与gid匹配、未过滤90000等统计数据的数据尾部，便于稍后与第4组的统计结果对比：

```
#3
      gid  cid  cname pwin0 pdraw0 plost0 pwin9 pdraw9 plost9
vwin0 ... gtid kwin kwinrq mplay mtid qj qr qs      tplay tweek
      28 512751 734 Smarkets 2.22 2.48 1.76 4.20 3.65 2.02
31.68 ... 905 0 -1 黑山 21 4928 0 0 3 2016-09-02 4
      29 512751 1304 Bet-52 2.92 3.31 2.20 3.55 3.38 1.92
31.16 ... 905 0 -1 黑山 21 4928 0 0 3 2016-09-02 4
      30 512751 90005 gavg 3.15 3.36 2.09 3.73 3.39 1.90
29.28 ... 905 0 -1 黑山 21 4928 0 0 3 2016-09-02 4
      31 512751 90009 gmax 4.36 3.76 2.30 4.66 3.75 2.25
33.11 ... 905 0 -1 黑山 21 4928 0 0 3 2016-09-02 4
      32 512751 90001 gmin 2.22 2.48 1.76 2.70 3.00 1.65
22.27 ... 905 0 -1 黑山 21 4928 0 0 3 2016-09-02 4

[5 rows x 35 columns]
```

第4组代码主要调用Pandas的内置统计函数，生成各种衍生数据，代码如下：

```
#
ds=pd.Series()
ksgn='pwin0'
```

```
df2[ksgn]=df2[ksgn].astype(float)
ds['xmin'],ds['xmax']=df2[ksgn].min(),df2[ksgn].max()
ds['xavg'],ds['xmed']=df2[ksgn].mean(),df2[ksgn].median()
#
ds['q-0.1']=df2[ksgn].quantile(0.1)
ds['q-0.25']=df2[ksgn].quantile(0.25)
ds['q-0.5']=df2[ksgn].quantile(0.5)
ds['q-0.75']=df2[ksgn].quantile(0.75)
ds['q-0.9']=df2[ksgn].quantile(0.9)
#
ds['xmad']=df2[ksgn].mad()
ds['var']=df2[ksgn].var()
ds['std']=df2[ksgn].std()
ds['skew']=df2[ksgn].skew()
ds['kurt']=df2[ksgn].kurt()
#
ds['cumsum']=df2[ksgn].cumsum()
ds['cummin']=df2[ksgn].cummin()
ds['cumprod']=df2[ksgn].cumprod()
#
ds['diff']=df2[ksgn].diff()
ds['pct_change']=df2[ksgn].pct_change()

print('\n#4')
print(ds)
```

第 4 组输出信息如下:

#4	
xmin	2.22
xmax	4.36
xavg	3.24633
xmed	3.1
q-0.1	2.895
q-0.25	2.9275
q-0.5	3.1
q-0.75	3.55
q-0.9	3.902

```

xmad                                0.378511
var                                0.224665
std                                0.473989
skew                                0.572162
kurt                                0.100644

cumsum      0      3.25
1      7.05
2      9.95
3     13.3...
cummin      0      3.25
1      3.25
2      2.90
3      2.90
4 ...
cumprod      0      3.250000e+00
1     1.235000e+01
2     3....
diff          0      NaN
1      0.55
2     -0.90
3      0.50
4 ...
pct_change   0      NaN
1      0.169231
2     -0.236842
3...
dtype: object

```

第4组输出数据较长，分别如下。

- 设置数据列 **pwin0** 即主队胜的开盘赔率，读者也可以自己修改。
- **xmin**、**xmax**、**xavg**、**xmed** 分别是最小值、最大值、平均值、中位数，其中最大值、最小值与 **dfk** 的数据相同，平均值有所差异，可能是计算方法不同。
- **q-0.1** 至 **q-0.9** 分别是各种分位数。
- **mad** 是根据均值计算平均绝对离差。

- `var` 和 `std` 分别为方差和标准差。
- `skew` 是样本值的偏度（三阶矩）。
- `kurt` 是样本值的峰度（四阶矩）。

以下函数返回的不是单一的数值，而是复合数据。

- `cumsum`，样本值的累计和。
- `cummin`、`cummax`，样本值的累计最小值和累计最大值。
- `cumprod`，样本值的累计积。
- `diff`，计算一阶差分（对时间序列很有用）。
- `pct_change`，计算百分数变化。

以上各种数值和函数的具体意义，请读者自行查阅有关资料，在此不再细谈。

5.2 科学计算

科学计算包括数值计算、可视化工具及数据交互，专业软件有 MATLAB、OCTAVE 和 Jupiter 等。

Python 语言在这方面的模块库的传统组合是：NumPy+SciPy+Matplotlib+iPython。

近年来，伴随新一代数据分析工具 Pandas 和交互式绘图神器 Plotly 的发布，目前笔者推荐的组合是：Pandas+Plotly+iPython。

传统的 NumPy 和 SciPy 等模块库并未消失，而是退居底层，下面对它们做一些简单的介绍。

- NumPy 模块库是 Python 语言最基本的科学计算工具包，最常用的是 N 维数组（矩阵、矢量）对象及优化的矩阵函数库，包括线性代数、傅里叶变换和随机数生成函数等。
- SciPy 模块库是一套开源的 Python 算法库和数学工具包，包含的模块有最优化计算、线性代数、积分、插值、快速傅里叶变换、信号处理和图像处理、微分方程求解等工程计算函数。其功能与软件 MATLAB、Scilab 和 OCTAVE 类似。Numpy 和 Scipy 通常配合使用，Python 中的大多数机器学习库都依赖于这两个模块。
- iPython 模块库是一个 Python 的交互式外壳，简单好用，功能也非常强大。支持语法高亮、代码调试、对象自省等功能，内置了许多很有用的功能和函数等，

非常容易使用。zwPython 的开发环境 Spyder 已经内置了 iPython 模块库，就在窗口的右下角。

5.3 人工智能

Python 人工智能软件包包括数据挖掘、机器学习、神经网络等模块库，主要用于数据建模，一般是使用 `scikit-learn` 模块库，再复杂一些就直接使用谷歌的 TensorFlow 开源架构，也有部分功能如人脸识别、车牌识别等偏重于图像分析的项目使用 OpenCV（图像识别、处理、图像数据采样）模块完成。

许多读者都对深度学习和神经网络很感兴趣，但是不知道从哪里开始学，也不知道使用哪种工具，下面的这些介绍想必可以为读者提供许多帮助。

目前，Python 语言常用的人工智能、机器学习模块库如下。

- **scikit-learn** 模块库。大名鼎鼎的 `scikit-learn` 是基于 NumPy、SciPy、Matplotlib 的开源机器学习工具包，主要涵盖分类、回归和聚类算法，例如 SVM、逻辑回归、朴素贝叶斯、随机森林和 k-means 等算法，代码和文档非常丰富，在许多 Python 项目中都有应用。例如，在 NLTK 语义分析模块库中，分类器方面就内置了 `scikit-learn` 接口，可以调用 `scikit-learn` 的分类算法及训练数据来设置分类器模型。
- **TensorFlow** 深度学习框架。TensorFlow 是谷歌的开源项目，目前在人工智能领域基本已经一统天下，是使用数据流图进行数值计算的开源库（这是所有神经网络固有的特征）。相对于其他传统人工智能软件，TensorFlow 的主要优点是功能全面、文档丰富，而且支持 GPU 与分布式计算。
- **Caffe** 深度学习框架。用 C++ 语言编写的老牌深度学习工具，由 Berkeley Vision and Learning Center (BVLC) 设计，采用模块化设计，速度极快，广泛应用于学术界和产业界，Caffe 内置 Python 和 MATLAB 开发接口。
- **Theano** 模块库。在 TensorFlow 问世以前，Theano 可以说是最著名的深度学习框架之一。其特点是：紧密集成 Numpy、高效的数据密集型 GPU 计算、高效的符号微分运算、高速和稳定的优化、动态生成 C 代码，以及广泛的单元测试和自我验证。Theano 使得构建深度学习模型更加容易，可以快速实现多种模型。
- **Lasagne** 模块库。Lasagne 是 Theano 中用于构建和训练网络的轻量级库，可以根

据 Theano 进行模块化的构建。简而言之，Lasagne 的功能是 Theano 的低级编程和 Keras 的高级抽象之间的一个折中。

- **Keras 模块库。**Keras 是最受欢迎的 Python 深度学习库，只需几行 Python 代码就可以构建深度学习架构。Keras 是一个最低限度的、模块化的神经网络库，可以把 Theano 和 TensorFlow 包装成更具人性化的 API。Keras 最主要的用户体验是：从构思到产生结果是一个非常迅速的过程。在 Keras 中，架构神经网络设计非常轻松，内置了大量人工智能相关优化算法。
- **Mxnet 模块库。**Mxnet 提供了多种语言接口，如 C++、Python、R、Java 等，有出色的分布式计算，它支持多个 CPU / GPU 机训练神经网络，甚至支持 AWS、Azure 云计算模式及 YARN 集群。
- **Sklearn-theano 模块库。**它的神奇之处就是可以把神经网络作为特征提取器，可用于评估一个特定的问题是否适合使用深度学习来解决，使机器学习变得更加容易。
- **Nolearn 模块库。**类似 Keras 模块库，Nolearn 也是 Theano 和 TensorFlow 的二次封装，API 接口更加简单，重要的是 Nolearn 中所有的代码都是与 scikit-learn 兼容的。
- **DIGITS 模块库。**DIGITS 采用 Python 开发的、基于 Web 的网络程序，用于辅助 Caffe 深度学习模块，简化 Caffe 编程。
- **MDP 模块库。**一个 Python 数据处理框架，内置了大量的监督学习算法、无监督学习算法和其他数据处理单元，很容易扩展。
- **PyBrain 模块库。**目标是为机器学习任务提供灵活、易用、强大的机器学习算法。功能包括神经网络、强化学习（及二者结合）、无监督学习和进化算法。PyBrain 以神经网络为核心，所有的训练方法都以神经网络为一个实例。
- **PyML 模块库。**有灵活的分类和回归架构，主要提供特征选择、模型选择、组合分类器和分类评估等功能。
- **Milk 模块库。**提供监督分类法与几种有效的分类分析，如 SVMs（基于 libsvm）、K-NN、随机森林和决策树。它还可以进行特征选择。这些分类可以在许多方面相结合，形成不同的分类系统。对于无监督学习，它提供 K-means 和 affinity propagation 聚类算法。
- **Monte 模块库。**Monte 是一套纯 Python 语言开发的机器学习库。它可以迅速构建神经网络、条件随机场、逻辑回归等模型，使用 inline-C 优化，极易使用和扩展。

- Pylearn2 模块库。Pylearn2 是基于 Theano、scikit-learn 的深度学习工具包，可以处理向量、图像、视频等数据，提供 MLP、RBM、SDA 等深度学习模型。

5.4 NLTK语义分析

语义分析环节主要是配合网络文本信息采集，如财经新闻、股票论坛、微博微信进行金融舆情分析，常用的语义分析模块有 NLTK、spaCy、Pattern 和 Gensim。此外，中文分词当中的 Jieba（结巴）模块库的应用也比较多。

语义分析包括语句切分和句法分析。语句切分提取文本中的基本单词，英文称为 tokenize（提取），中文称为“中文分词”，然后进行词性标注。句法分析是关键词提取、文本分类、情感分析等。有了这些最基本的语义数据后，就可以进行专业的统计分析和神经网络分析。

目前，常用的 Python 语义分析模块库如下。

- NLTK 模块库，是 Python 处理语言数据的领先平台。它为如 WordNet 这样的词汇资源提供了简便、易用的界面。它还具有为文本分类（classification）、文本标记（tokenization）、词干提取（stemming）、词性标记（tagging）、语义分析（parsing）和语义推理（semantic reasoning）准备的文本处理库。配套文档有：《用 Python 进行自然语言处理》中文版，以及更加专业的《Python Text Processing with NLTK 2.0 Cookbook》，内容包括 NLTK 代码结构、语料库和语义分析模型设计等。
- spaCy 模块库，是一套商业化开源软件，是使用 Python 和 Cython 进行工业级自然语言处理的软件。底层采用 C 语言优化，使用 Python 接口，是目前最快的和水平最高的自然语言处理工具。
- Pattern 模块库，用于词性标注（part-of-speech taggers）、n-gram 搜索、情感分析、单词提取、词性标注、句子切分、语法检查、拼写纠错和句法分析等。它还支持矢量空间建模、聚类分析及向量机。Pattern 由比利时安特卫普大学 CLiPS 实验室出品，不仅是一套文本处理工具，而且是一套 Web 数据挖掘工具，内置多种数据抓取模块（包括 Google、Twitter、维基百科的 API，以及爬虫和 HTML 分析器）、文本处理模块（词性标注、情感分析等）、机器学习模块（VSM、聚类、SVM）及可视化模块等。
- TextBlob 模块库，是处理文本数据的一个 Python 库。它为深入挖掘常规自然语

言处理提供简单易用的 API。它其实是基于对 NLKT 和 Pattern 模块库的二次封装，同时提供了很多文本处理功能的接口，包括词性标注、名词短语提取、情感分析、文本分类、拼写检查等，甚至包括翻译和语言检测。

- MBSP 模块库，与 Pattern 同源，同出自比利时安特卫普大学 CLiPS 实验室，提供了单词提取、句子切分、词性标注、句法分析等基本的文本处理功能。
- Gensim 模块库，专业的主题模型 Python 工具包，代码和文档非常漂亮，可用于主题建模、文档索引及使用大规模语料数据的相似性检索。相比于 RAM，它能处理更多的输入数据。作者称它是：“根据纯文本进行非监督性建模最健壮、最有效的、最让人放心的软件”。
- PyNLPL 模块库，全名是 Python Natural Language Processing Library，是一个用于自然语言处理的 Python 库。它由一系列的相互独立或相互松散独立的模块构成，用于处理常规或不太常规的 NLP 任务。PyNLPL 可用于 n-gram 计算、频率列表和分布、语言建模。除此之外，还有更加复杂的数据模型如优先级队列，还有搜索引擎如波束搜索。
- Polyglot 模块库，是一个支持海量多语言的自然语言处理工具。它支持多达 165 种语言的文本标记、196 种语言的语言检测、40 种语言的命名实体识别、16 种语言的词性标注、136 种语言的情感分析、137 种语言的字根嵌入、135 种语言的形态分析及 69 种语言的音译。
- MontyLingua 模块库，是一个免费的、常识丰富的、端对端的英语自然语言理解软件。用户只需要将原始英文文本输入 MontyLingua，就能输出文本的语义解释。该软件完美适用于信息提取、需求处理及问答。根据给定的英语文本，它能提取主语、动词、形容词、名词短语和动词短语，并提取人的名字、地点、事件、日期和时间，以及其他语义信息。
- langid.py 模块库，是专业的语言检测模块库，目前支持 97 种语言的检测，提供了很多易用的功能，可定制训练自己的语言检测模型。
- Jieba 模块库，“结巴”中文分词，功能包括支持三种分词模式（精确模式、全模式、搜索引擎模式）、支持繁体分词、支持自定义词典等，是目前一个非常不错的 Python 中文分词解决方案。
- xTAS 模块库，国内新推出的 Python 文本分析模块库，基于 Celery 和 Elasticsearch 全文搜索引擎。

5.5 数据清洗统计分析

除了 Pandas 数据分析软件外，常用的 Python 数据分析、统计模块库还有：scipy.stats、statsmodels 和 statistics 等。

- **scipy.stats** 统计模块库，比较基础，主要围绕随机变量提供数值方法，比如随机变量的分位数/cdf、构造分布、分布对象模型。还有一些检验方法，例如 KS 检验、T 检验、正态性检验和卡方检验，以及一些不成体系的估计方法（随机变量也有估计方法）。scipy.stats 缺乏最重要的回归统计方法体系，这个完全由 statsmodels 提供，scipy.stats 主要围绕回归模型提供操作方法，如数据访问方式、拟合、绘图和报告诊断等。
- **statsmodels** 统计模块库，是 Python 的统计建模和计量经济学工具包，scipy.stats 以前有一个 models 子模块，后来被移除了，这个模块被重写并成为现在独立的 statsmodels 模块库。statsmodels 模块库包括描述统计、统计模型估计和推断、统计测试和统计数据挖掘等功能，每个模型都会生成一个对应的统计结果。统计结果会和现有的统计包进行对比来保证其正确性。
- **statistics** 统计模块库，用 C 语言写的 Python 统计扩展模块，主要功能有概率密度函数、随机变量分布、均值、中值、皮尔森相关、函数拟合和线性回归线等。
- **PyMVPA** 模块库，是为大数据集提供统计学习分析的 Python 工具包，它提供了一个灵活、可扩展的框架。它提供的功能有分类、回归、特征选择、数据导入导出和可视化等。

5.6 数据可视化

Python 数据分析的一个关键是数据可视化，无论是传统的柱形图、扇形图，还是现在的互动策略分析，都需要大量的可视化图表分析，具体到编程代码，就是各种 Python 绘图模块库。

Python 图像处理主要是像素编辑，这方面常用的模块库是 Pillow 与 OpenCV，与数据可视化关联不大。

Python 常见的绘图模块库如下。

- Matplotlib, 是最常用的绘制二维图形的 Python 模块库, 非常强大, 也很复杂。它提供许多 MATLAB 的绘图函数, 可以绘制多种图形, 如线图、直方图、饼图、散点图及误差线图; 可以比较方便地定制图形的各种属性, 如图形的类型、颜色、粗细、字体的大小等; 它能够很好地支持一部分 TeX 排版命令, 可以比较美观地显示图形中的数学公式。
- Seaborn, 基于 Matplotlib, 是更加简单、强大的 Python 绘图模块库, 提供更加现代的色彩组合与图表形式, 并针对数据统计进行了大量的优化。Seaborn 图表模块库使数据看起来更具有吸引力, 还可以简单地创建更复杂的图表, 也可以和 Pandas 集成。
- ggplot, 最早在 R 语言中实现, 目前也有 Python 版本。ggplot 与 Seaborn 相似, 也是基于 Matplotlib 的, 用于简化 Matplotlib 可视化, 并改善可视化效果。使用 ggplot 绘图的过程就是选择合适的几何对象、图形属性和统计变换来充分揭露数据中所含有的信息的过程。ggplot2 采用特殊的图形语法结构, 需要一定的时间去学习, 但是当掌握该图形语法的时候, 就会感受到其中的优雅。
- Bokeh, 是一个专门针对 Web 浏览器呈现功能的交互式可视化 Python 库, 这是 Bokeh 与其他可视化库最核心的区别。
- Pychart, 用于创建高品质封装的 PostScript, 是 PDF、PNG 或 SVG 图表 Python 库。
- PLPlot, 用于创建科学图表的跨平台软件包。以 C 类库为核心, 支持各种语言绑定 (C、C++、FORTRAN、Java、Python、Perl 等)。
- Vpython, 是 Visual Python 的简写, 是由 Carnegie Mellon University 在校学生 David Scherer 于 2000 年撰写的一个 Python 3D 绘图模块。
- pygal, 可用来创建 SVG 图表, SVG 文件对于创建交互图表非常有用, 可以很容易地创建个性化的、视觉体验很好的图表, 不过没有基于 Matplotlib 的方案那样灵活。
- Plotly, 新一代互动型量化绘图库。

Python 绘图模块库数不胜数, 其中称为经典的也有数十个。目前, Plotly 作为新一代互动型绘图库, 基本上可以说一统天下。如图 5-3 所示是 Plotly 绘图模块库部分内置的图形类型。

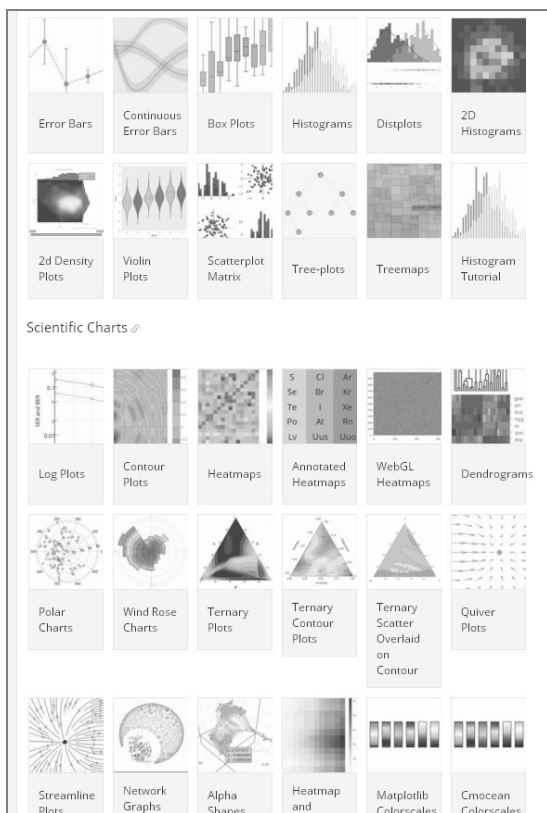


图 5-3 Plotly 绘图模块库的部分内置图形类型

目前，Plotly 绘图模块库支持的图表类型和图表工具如下。

- 基本图表：20 种。
- 统计和海运方式图：12 种。
- 科学图表：21 种。
- 财务图表：2 种。
- 地图：8 种。
- 3D 图表：19 种。
- 报告生成：4 种。
- 连接数据库：7 种。
- 拟合工具：3 种。
- 实时图表：4 种。

当然，这些图表类型和图表工具并非一成不变，随着 Plotly 的不断升级，未来支持的图表类型肯定会越来越多。

原本 Plotly 是收费的商业软件，幸运的是，2016 年 6 月，Plotly 绘图模块库提供了免费的社区版本，并新增了 Python 等多种编程语言的接口，以及离线模式支持，这对于我国广大用户而言，提供了现实的技术支持。

笔者在编著本书时，特别针对多种 Python 绘图模块库进行了测试，测试结果表明，Plotly 绘图模块库的确不愧是新一代 Python 绘图模块的王者之选，也是各种 Web 平台的优先选择对象。

以笔者个人的测试感觉而言，Plotly 绘图模块既有 Matplotlib 绘图模块的强大与灵活，也有 Seaborn 统计绘图的现代配色组合与优雅报表。

与传统绘图模块不同，Plotly 绘图模块可直接生成 PNG 等图像文件，生成的是一个内置 JavaScript 脚本的 HTML 网页文件，虽然文件体积比 bokeh 绘图模块生成的文件体积略大，但互动性方面强大得多。

在 3D 图表方面，虽然笔者没有实际测试，但基于 Plotly 网站的案例可以看出，相比传统 Python 3D 图表模块库，无论在图表种类格式，还是色彩组合方面，Python 都显得更加灵活，也更加强大。

如图 5-4 所示是 Plotly 的效果截图，虽然是静止的，但在浏览器中是 100%JavaScript 脚本的函数图形，支持各种互动功能，读者可以好好体会一下，并且看看网页源码。

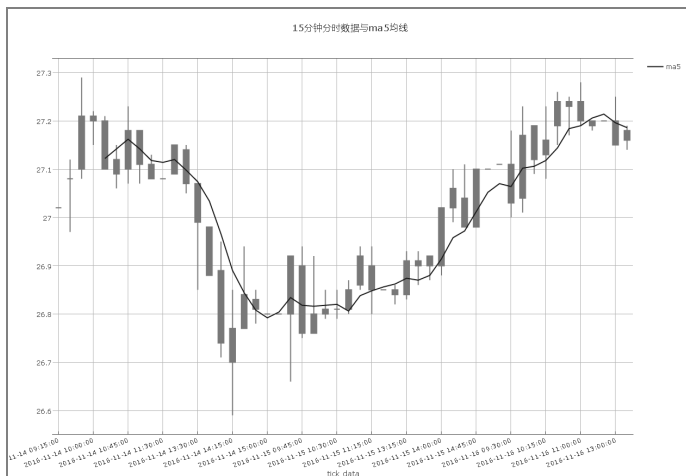


图 5-4 Plotly 效果截图

Plotly 原本是基于 JavaScript 的数据图表分析绘制模块库，在编程的灵活性和图表的丰富性方面非常强大，优点数不胜数。

- Plotly 本身是一款独立的 Web 数据可视化工具，界面友好，提供强大的互动性操作。
- 基于现代的配色组合、图表形式，比起 Matplotlib、R 语言的图表，更加现代、绚丽。
- 简单强大的 3D 图表绘制功能，支持多种格式。
- 对图形参数的修改十分简单直观，便于初学者使用。
- 有 Python、R、MATLAB、Jupyter、Excel 等多种版本的接口。
- 与 Pandas 数据分析软件无缝集成，并提供了专门的 Plotly 绘图模块库，设计的图表非常吸引人，而且高度互动。得益于其完善的文档和简单的 Python API，起步入门也很容易。

6

第 6 章

辅助工具

6.1 性能优化

Python 是动态语言，开发效率非常高，运行效率相对 C 语言有不少差距。在工程领域，为 Python 加速的方法有很多，比如 PyPy、Cython、Numba、Numexpr 和 LLVM 等。

数据分析需要大量的实时计算，属于计算密集型项目，通常需要采用 GPU 加速或者计算机集群等优化手段，做高频交易的甚至有采用 FPGA 硬件加速方案的。

GPU 加速或者计算机集群都属于非常专业的领域，一般用户无需了解细节，直接使用就可以了，在此不再做深入介绍。

对于 Python 数据分析，最基本的技巧如下。

- 少用 for 循环，尽量用 NumPY/Pandas 的向量化方法。必须使用 for 循环时，用 Numba 加速方案。
- 少用自定义函数，先看看 NumPY/Pandas 是不是已有现成的功能。
- 使用 Numpy 的加速包，比如 Numexpr。
- 安装 Intel MKL 优化工程函数库。
- 关键部分如交易接口，用 C/C++实现。

下面介绍 Python 常用的性能优化模块库、技术手段和编程技巧。

Python 性能优化方案很多，主要使用的模块库有：Numexpr、Numba、PyPy、mtrd、CUDA 和 Blaze。

6.1.1 Numexpr 矢量加速库

Numexpr 是一个 Python 加速库，采用优化的矢量数组运算虚拟机技术，是基于多核与大内存的加速方案，因为采用了类似 zwQuant 的数据预处理计算，通过查表简化计算过程，所以在计算数组时，甚至比采用 C 语言编写的、高度优化的 Numpy 运算函数还要快 5~10 倍，Numexpr 的输入是一个字符串表达式。

6.1.2 Numba 支持 GPU 的加速模块库

Numba 模块库是基于 LLVM 的 Python 加速模块库，其最大的优势在于支持 GPU 加速，其商务版本 NumbaPro 是 NVIDIA 公司唯一认可的 Python 语言 GPU 加速方案。

根据测评结果，Numba 的 CPU 加速效果与 Cython 差不多，但是使用非常简单，无需修改 Python 源码，可以通过 @jit 函数修饰符来实现。

Numba 以 GPU 或多核 CPU 为目标编译代码，实现方式同样十分简单。

6.1.3 Blaze 大数据优化模块库

Blaze 模块库是高效处理大数据的 Python 模块库，专为大数据打造用于处理分布式各种数据源的计算。它扩展现有的数学计算库 Numpy 和科学计算库 Scipy，使其更适应大数据库技术。Blaze 聚焦在内核外处理超过系统内存容量的大型数据集，并同时支持分布式数据和流数据。

Blaze 模块库整合了包括 Python 的 Pandas、Numpy、SQL、Mongo、Spark 在内的多种技术，使用 Blaze 能够非常容易地与一个新技术进行交互。

Blaze 主要提供直观的各种工具访问来展现性能。Blaze 忽略用户使用的各种不同的计算方案 and 不同类型的数据库，为用户提供统一、友好、熟悉的界面。它帮助

用户更好地与文档、数据结构及数据库进行交互，根据需要适当优化和转换用户查询语句以提供一个流畅和交互式会话。

6.1.4 Pyston 加速模块

Pyston 加速方案由 DropBox 公司负责开发，其原理与 PyPy 类似，也是把 Python 转换为 C 语言，用 LLVM 优化编译后运行，速度比 CPython 快，但目前还赶不上 PyPy。

Pyston 对于 C 语言的支持比 PyPy 好，而且 Python 语言的设计师吉多·范罗苏姆（Guido van Rossum）离开谷歌公司后，加盟的公司就是 DropBox。

6.1.5 PyPy 加速模块

PyPy 表示“用 Python 实现的 Python”，是用 Rpython（CPython 的子集）实现的 Python，将 Python 代码转换成 C 语言代码，同时在转换的过程中，进行代码优化，如 Just-in-Time(JIT)技术，能让 Python（事实上是转换后的目标代码）的执行速度更快。

根据官网的基准测试数据，PyPy 比 CPython 实现的 Python 要快 6 倍以上。快的原因是使用了 Just-in-Time (JIT) 编译器，即动态编译器，与静态编译器（如 gcc、javac 等）不同，它利用程序运行的过程对数据进行优化。

6.1.6 Cython

Cython 是 Python 的一个超集，允许调用 C 函数及声明变量来提高性能。Cython 代码与 Python 不同，必须先编译。Cython 里可以载入 Python 扩展（例如 `import math`），也可以载入 C 的库的头文件（例如 `cdef extern from "math.h"`），另外也可以用它来写 Python 代码。

Cython 本质上是另一个不再开发的类似类库 Pyrex 的分支，它将类 Python 代码编译成 C 库，使其可以在一个 Python 文件中调用。

6.1.7 其他优化技巧

- 使用最新版本的 Python，虽然 Python 2.7 比 Python 3.x 在某些环节速度略快，不过这种速度优势是以牺牲大量新功能为代价的。而且 Python 3.5 以后，伴随着 Python 3 的普及，性能优化也开始加速，在 Python 3.5 中，进行了大量的性能优化升级，例如支持矩阵运算符号、用 C 语言重写 `collections.OrderedDict` 模块，速度提高了 4~100 倍等。
- 合理使用 `copy` 与 `deepcopy`，对于 `dict` 和 `list` 等数据结构的对象，`copy` 使用的是引用的方式，`deepcopy` 使用的是递归复制的方式，速度慢十倍。
- 使用 `dict` 或 `set` 查找元素时，Python 语言 `dict` 和 `set` 都是使用哈希表来实现的，比其他方式，如列表、字符串，快一个数量级。
- 使用 `join` 连接字符串比使用普通的“+”（加号）连接字符串在性能上快五倍。这是 Python 的一个 bug，不够人性化，笔者认为未来的版本升级会在语言内部进行优化。
- 优化循环，尽量减少循环过程中的计算量，有多重循环的尽量将内层的计算提到上一层。
- 充分利用 `lazy if-evaluation`（懒惰结算）的特性，Python 中条件表达式是 `lazy evaluation` 的，也就是说如果存在条件表达式 `if x and y`，在 `x` 为 `false` 的情况下，`y` 表达式的值将不再计算。因此，可以利用该特性在一定程度上提高程序效率。

6.2 网页信息抓取

网页信息抓取其实包含以下三个意思。

- **Get**，获取下载网页信息、图像数据、赔率数据。
- **Parse**，解析 HTML 等网页信息，提取有效的文本信息、网页链接，过滤垃圾信息等。
- **Post**，提交并发布自己的数据信息。

对于足彩数据分析而言，抓取网页数据主要是用于情感分析、足彩数据的舆情分析，特别是配合机器学习、人工智能分析。此外，足彩比赛数据和赔率数据也需要读者自己动手通过网页抓取提交数据。

Python 提供了一些很不错的网页爬虫工具框架，既能爬取数据，又能获取和清洗数据，常用的模块库参见后续介绍。

6.2.1 Requests 人性化的网络模块

Requests 是 Python 官方推荐的 HTTP 网络编程模块库。Python 标准库中的 Urllib2 模块虽然也可以完成大多数 HTTP 网络编程功能，但是它的 API 接口不友好，非常复杂，需要进行大量的工作，甚至包括各种方法覆盖来完成最简单的任务。

尽管可以通过标准库中的 Urllib2 模块来操作网页表单，但有些时候更友好的代码可以让工作更轻松。

Requests 模块库在处理复杂 HTTP 命令、Cookies 等方面更为高效，Requests 模块库的设计师 Kenneth Reitz 对 Python 核心工具的评价是：

Python 标准 Urllib2 模块提供了用户所需要的绝大多数 HTTP 功能，但是其 API 却很差。它的创建是用于不同时间及不同网页的，即使执行一个简单的任务也需要大量的工作（甚至重写方法等）。

事情并不应该如此，至少在 Python 中不应如此。

Requests 模块库简洁好用，其宣传口号就是：让 HTTP 服务人类。

Requests 模块库允许用户发送人性化的 HTTP 请求命令，无需手动为 URL 添加查询字符串，也不需要 POST 数据进行表单编码。Keep-alive 和 HTTP 连接池的功能是 100%自动化的，由 Requests 模块库内部的 Urllib3 子模块完成。

下面以常见的网站模拟登录案例来说明 Requests 模块库的基本功能，代码很简单，只有几行：

```
rq = requests.session()
data = {'user': '用户名', 'passwd': '密码'}
#post 换成登录的地址,
res=rq.post('http://www.xxx.com/index.php?action=login',data);
#换成抓取的地址
htm=rq.get('http://www.xxx.com/archives/101/');
```

读者看到了，只用 4 行代码即可完成全部的模拟登录工作。

6.2.2 Scrapy 网页爬虫框架

网页爬虫是在网页中抓取数据的程序，使用它能够抓取特定网页的 HTML 数据。虽然可以利用模块库开发一个爬虫程序，但是使用框架可以大大提高效率，缩短开

发时间。Scrapy 是一个使用 Python 编写的、轻量级的网页爬虫框架，简单轻巧，并且使用起来非常的方便。

从严格意义上来说，Scrapy 已经不是一个网络模块库，而是一套完整的网页爬虫框架，或者说是一套网站下载系统，它可以把一些中小型网站完整地抓取下来。

Scrapy 是使用 Python 语言开发的一个快速、高层次的网页抓取框架，用于抓取 Web 站点并从页面中提取结构化的数据。Scrapy 用途广泛，可以用于数据挖掘、监测和自动化测试。

Scrapy 吸引人的地方在于它是一个框架，任何人都可以根据需求方便地修改。它也提供了多种类型爬虫的基类，如 BaseSpider、Sitemap 等，最新版本又提供了 Web 2.0 爬虫的支持。

如图 6-1 所示是 Scrapy 的内部结构图，如图 6-2 所示是 Scrapy 网站首页截图。

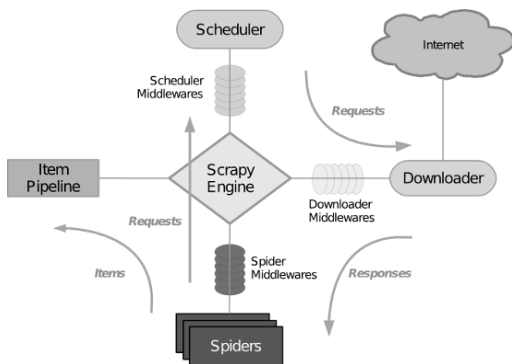


图 6-1 Scrapy 内部结构图



图 6-2 Scrapy 网站首页截图

Scrapy 的官方网址：<http://scrapy.org/>。

GitHub 项目网址：<https://github.com/scrapy/scrapy>。

6.2.3 Beautiful Soup 4

Beautiful Soup 的意思是“可口的汤羹”、“甲鱼汤”，它是一套 Python 模块库，用于网页的文本分析、数据清洗，目前已经发展到第 4 代。

Beautiful Soup 官方网址是 <http://www.crummy.com/software/BeautifulSoup/>，其网站截图如图 6-3 所示。



图 6-3 Beautiful Soup 网站截图

6.3 其他工具模块

6.3.1 Logging 日志模块

Java 语言中最通用的日志模块莫过于 Log4j 了,Python 语言中也自带了 Logging 模块,该模块的用法其实和 Log4j 类似。

Python 语言中的 Logging 模块与 Log4j 的机制是一样的,只是具体的实现细节不同。Logging 日志模块主要涉及以下四个类。

- **Logger**, 提供日志接口,供应用代码使用。
- **Handler**, 将日志记录(log record)发送到合适的目的地,例如文件、Socket 等。
- **filter**, 提供一种优雅的方式来决定输出哪条日志记录。
- **formatter**, 指定日志记录输出的具体格式。

Python 语言中的 Logging 模块提供了通用的日志系统,可以方便第三方模块或者应用使用。这个模块提供不同的日志级别,可以采用不同的方式记录日志,比如文件、HTTP GET/POST、SMTP 和 Socket 等,甚至可以自己实现具体的日志记录方式。

6.3.2 Debug 调试工具

软件工程研究表明，只有不到 1% 的程序能够一次完成并正常运行，实际编程过程中总会有各种各样的 bug 需要修正。有的 bug 很简单，看错误信息就知道了；有的 bug 很复杂，我们需要知道出错时哪些变量的值是正确的，哪些变量的值是错误的。因此，需要一整套调试程序来修复 bug。

- **Print 大法**。这个调试手段简单、直接、粗暴，但非常有效，就是把可能有问题的变量打印出来看看，这也是大部分 Python 用户最常用的调试手段。之所以流行，是因为无需任何第三方模块，直接打印看变量参数，一般 80% 的问题都能够解决。用这种模式最大的缺点是将来还得删掉它，不然运行结果中会包含很多垃圾信息，笔者一般是在调试完成后，在 Print 前面加上注释符号进行屏蔽。
- **日志**。一般的开发环境都有内置的 log 窗口，当然也可以自己通过日志模块设置调试信息。使用日志方式的最大好处是，可以记载 bug 出现的具体时间和语句位置，而且便于第三方远程技术支持。
- **pdb 调试模块库**。一般的 Python 开发环境都内置了该模块库，或者更加强大的类似工具。pdb 是 Python debugger 的简称，为 Python 程序提供了一种交互的源代码调试功能，主要特性包括设置断点、单步调试、进入函数调试、查看当前代码、查看栈片段和动态改变变量的值等。
- **trace 模块**。与 pdb 模块类似，一般的开发环境都已经内置，trace 模块可以监控 Python 执行程序的方式，用来研究程序调用图，进而发现模块之间的依赖关系，使调试工作变得更加简单，对于 traceback 错误信息，可以输出到日志文件或消息对话框中。
- **性能测试**。一般使用 profile 和 timit 模块来测试程序的速度，找出程序中到底是哪里很慢，进而把这部分代码独立出来进行调优。

6.3.3 re 正则表达式

正则表达式是处理字符串、文本信息的神器，也是 Python、C#、Java 等现代编程语言与 C、Pascal、FORTRAN 等传统编程语言的一个标志性差别。

正则表达式，又称正规表示式、正规表示法、正规表达式、规则表达式、常规

表示法（英文是：regular expression，在代码中常简写为 regex、regexp 或 re）。正则表达式使用单个字符串来描述、匹配一系列匹配某个句法规则的字符串。在很多文本编辑器里，正则表达式通常被用来检索、替换那些匹配某个模式的文本。

许多程序设计语言都支持利用正则表达式进行字符串操作，正则表达式的概念最初是由 UNIX 中的工具软件（例如 sed 和 grep）普及的。

正则表达式原本并不是 Python 语言标准库，Python 语言自 1.5 版本起增加了 re 模块，它提供 Perl 风格的正则表达式模式，re 模块使 Python 语言拥有全部的正则表达式功能。

正则表达式是用于处理字符串的强大工具，拥有自己独特的语法及一个独立的处理引擎，效率上可能不及字符串自带的方法，但功能十分强大。

如果想了解更多关于正则表达式的内容，re 模块的 Python 官方文档是一个非常好的资源。

在后面的章节里，我们将进一步讨论 Python 语言的正则表达式应用，例如对象匹配、字符串替换，以及从网页文件中提取文本信息。

6.3.4 并行编程

GIL（全局解释器锁）是 Python 语言争议最大的地方之一。

Python 代码的执行由 Python 虚拟机（也叫解释器主循环，CPython 版本）来控制，Python 在设计之初就考虑到在解释器的循环中只有一个线程在执行，即在任意时刻，只有一个线程在解释器中运行。对 Python 虚拟机的访问由全局解释器锁（GIL）来控制，正是这个锁能保证同一时刻只有一个线程在运行。

Python 因为 GIL 的存在，很难充分利用多核 CPU 的优势。但是，可以通过内置的模块 multiprocessing 实现下面几种并行模式。

- 多进程：对于 CPU 密集型的程序，可以使用 multiprocessing 的 Process、Pool 等封装好的类，通过多进程的方式实现并行计算。但是因为进程中的通信成本比较大，所以对于进程之间需要大量数据交互的程序，效率未必有大的提高。
- 多线程：对于 IO 密集型的程序，multiprocessing.dummy 模块使用 multiprocessing 的接口封装线程，使得多线程编程变得非常轻松，比如可以使用 Pool 的 map 接口，简捷高效。

- 分布式: multiprocessing 中的 Managers 类提供了可以在不同进程之间共享数据的方式, 可以在此基础上开发出分布式的程序。

不同的业务场景可以选择其中的一种或几种的组合实现程序性能的优化。

6.4 网络辅助资源

大数据是目前全球最热门的产业领域之一, 虽然本书介绍了不少相关的 Python 模块库, 但随着行业的发展, 每天都有新的软件问世, 同时, 现有的软件与模块库也在不断地升级优化。为此, 笔者特意介绍几个常用的 Python 网络资源站点, 便于读者扩充资源。

根据笔者的经验, 与 Python 数据分析有关的资源网站和社区如下。

- <https://pypi.python.org/pypi>, 如图 6-4 所示为 PyPI 网站截图, 该网站是 Python 语言官方模块库下载中心, 目前有近三万个模块库。
- <http://www.lfd.uci.edu/~gohlke/Pythonlibs/>, 如图 6-5 所示为 LFD 网站截图, 该网站是 LFD 二进制 Python 模块库下载网站。
- <https://github.com>, 如图 6-6 所示为 GitHub 网站截图, 该网站是全球最大的开源项目网站。
- <http://stackoverflow.com>, 如图 6-7 所示为 stack overflow 网站截图, 该网站是全球最大的 IT 技术问答网站。
- <http://pydata.org>, 如图 6-8 所示为 PyData 网站项目截图, 该网站是目前 Python 领域数据分析最活跃的网站, 由 PyData 团队维护。
- <http://scikit-learn.org>, 如图 6-9 所示是 scikit-learn 网站截图, 该网站是老牌人工智能模块库网站, 有大量相关的技术文档资源。
- <http://ziwang.com> 或 <http://TopQuant.vip>, 如图 6-10 所示为 TopQuant 极宽量化社区 (原 zw 量化) 网站截图, 是我国第一家专业 Python 量化社区。
- <http://www.cnblogs.com>, 如图 6-11 所示为博客园网站截图, 该网站是我国最大的程序员博客网站。
- <http://wiki.mbalib.com>, 如图 1-12 所示为 MBA 智库百科网站截图, 该网站是全球最大的中文经管百科网站。
- <https://www.zhihu.com>, 如图 6-13 所示为知乎网站截图。



图 6-4 PyPi 网站截图

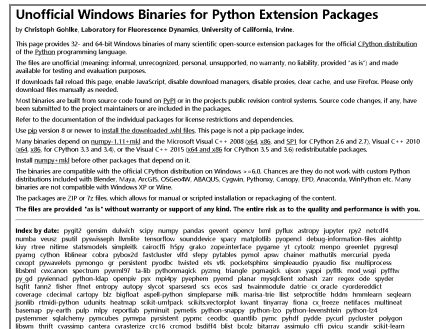


图 6-5 LFD 网站截图



图 6-6 GitHub 网站截图

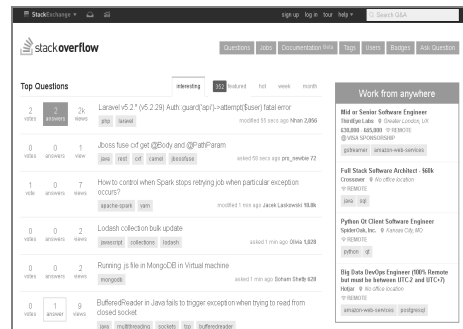


图 6-7 stack overflow 网站截图

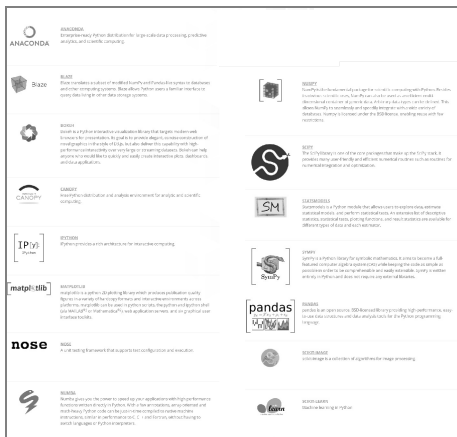


图 6-8 PyData 网站项目截图

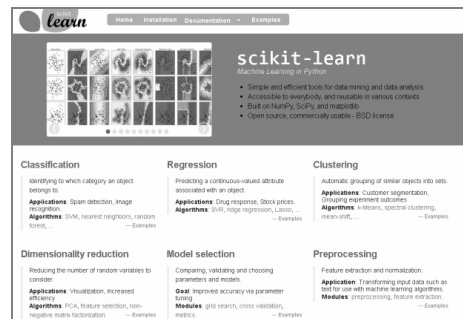


图 6-9 scikit-learn 网站截图



图 6-10 TopQuant 极宽量化社区网站截图



图 6-11 博客园网站截图



图 6-12 MBA 智库百科网站截图



图 6-13 知乎网站截图

6.5 arrow优雅简捷的时间模块库

前面简单介绍了 Python 数据分析常用的一些模块库，本节具体讲解 arrow 时间模块库。

arrow 是一个轻量级的 Python 时间模块库，提供了更加易懂和友好的方法来创建、操作、格式化和转换日期、时间和时间戳。它增强了 datetime 类型，完善了时间函数，通过提供友好的 API 接口来满足一般的场景创建。简单地说，它让用户使用更少的输入、更短的代码来完成时间处理工作。arrow 的灵感大部分来源于 moment.js 和 Requests。

Python 虽然有不少的日期、时间和时区的模块库，但从可用性的角度来看，它们并不是很完美，方法太复杂，不能高效地应用到工作中。

- 太多的模块：datetime、time、calendar、dateutil 和 pytz 等。
- 太多的类型：ate、time、datetime、tzinfo、timedelta 和 relativedelta 等。
- 时区和时间戳格式转换起来非常麻烦，还要计算转换。
- 功能缺陷：ISO-8601 解析、时间跨度不够人性化等。

arrow 时间模块库拥有以下特点。

- 兼容 datetime 模块。
- 支持 Python 2.6、Python 2.7 和 Python 3.x。
- 可识别时区，默认是 utc。
- 在很多常见的使用场景中提供了超级简单的创建选项。
- 升级了.replace 方法，支持相对位移，包括周。
- 对字符串提供了格式化和解析，包括 ISO-8601 格式的字符串自动解析。
- 时区转换非常方便、直接。
- 简单的时间戳操作，将时间戳作为一个属性。
- 从年到微秒，所有和时间相关的类型支持时间跨度、范围、向下取整和取上取整。
- 支持一个不断扩充的区域名单。
- 提供人性化的时间输出信息，例如两个月前、圣诞节前几日等，而且支持多种语言，包括中文。
- 可以创建简洁、智能的接口。
- 可以轻松更换和改变属性。
- 丰富的解析和格式化选项。
- 可扩展的工厂模式，支持自定义 arrow 派生类型。

Arrow 官方网址：<http://crsmithdev.com/arrow/>。

GitHub 项目网址：<https://github.com/crsmithdev/arrow>。

6.5.1 案例 6-1：arrow 入门案例

下面通过实际的案例来介绍 arrow 时间模块库的使用。

案例 6-1 的文件名是 zc601_arrow.py，主要代码如下：

```
#-----code
```

```

utc = arrow.utcnow()
print('utc,',utc)
utc = utc.replace(hours=-1)
print('utc2,',utc)
#
xtim=arrow.get('2013-05-11T21:23:58.970460+00:00')
print('xtim,',xtim)
print('xtim2,',xtim)

#
print('')
local = utc.to('US/Pacific')
print('local,',local)
print('local.timestamp,',local.timestamp)
print('localformat()',',',local.format())
print('localformat("YYYY-MM-DD HH:mm:ss
ZZ"),',local.format('YYYY-MM-DD HH:mm:ss ZZ'))
print('')
print('local.humanize()',',',local.humanize())
print('fr,',local.humanize(locale='fr_fr'))
print('cn,',local.humanize(locale='zh_cn'))

```

案例 6-1 的运行结果如下：

```

utc, 2016-12-28T09:04:42.064458+00:00
utc2, 2016-12-28T08:04:42.064458+00:00
xtim, 2013-05-11T21:23:58.970460+00:00
xtim2, 2013-05-11T21:23:58.970460+00:00

local, 2016-12-28T00:04:42.064458-08:00
local.timestamp, 1482912282
localformat(), 2016-12-28 00:04:42-08:00
localformat("YYYY-MM-DD HH:mm:ss ZZ"), 2016-12-28 00:04:42 -08:00

local.humanize(), an hour ago
fr, il y a une heure
cn, 1 小时前

```

前面的输出信息与传统的时间模块输出信息差不多，需要注意的是，`utc2` 的输出数据是通过 `replace` 替换命令完成的，也可以用 `shift` 移动命令。

```
utc = utc.replace(hours=-1)
```

`arrow` 时间模块库有趣的地方在于最后几行的输出信息，非常人性化，而且支持多种语言，最后一行是中文输出，倒数第二行是法文输出。

`arrow` 时间模块库支持数十种语言，这些功能源码都在其子模块 `locales.py` 中，读者也可以自己扩充。

6.5.2 创建 `arrow` 时间对象

创建 `arrow` 时间对象很简单，如获取当前时间的代码是：

```
xtim1= arrow.utcnow()
xtim2= arrow.now()
xtim3=arrow.now('US/Pacific')
```

`xtim1` 的 `utcnow` 函数返回的是 `utc` 世界时间，`xtim2` 中的 `now` 函数返回的是标准时间，`xtim3` 在 `now` 函数中设置了时区，所以返回的是美国太平洋时间。

6.5.3 创建时间戳

`arrow` 模块库可以根据整数或浮点数甚至字符串，获取时间戳数据，读者可以参看以下代码：

```
>>> arrow.get(1367900664)
<Arrow [2013-05-07T04:24:24+00:00]>

>>> arrow.get('1367900664')
<Arrow [2013-05-07T04:24:24+00:00]>

>>> arrow.get(1367900664.152325)
<Arrow [2013-05-07T04:24:24.152325+00:00]>

>>> arrow.get('1367900664.152325')
<Arrow [2013-05-07T04:24:24.152325+00:00]>
```

使用 `get` 函数处理时间，可以指定一个时区处理日期和时间：

```
>>> arrow.get(datetime.utcnow())
<Arrow [2013-05-07T04:24:24.152325+00:00]>
```

```
>>> arrow.get(datetime.now(), 'US/Pacific')
<Arrow [2013-05-06T21:24:32.736373-07:00]>

>>> from dateutil import tz
>>> arrow.get(datetime.now(), tz.gettz('US/Pacific'))
<Arrow [2013-05-06T21:24:41.129262-07:00]>

>>> arrow.get(datetime.now(tz.gettz('US/Pacific'))
<Arrow [2013-05-06T21:24:49.552236-07:00]>
```

以下代码是从字符串解析时间数据：

```
>>> arrow.get('2013-05-05 12:30:45', 'YYYY-MM-DD HH:mm:ss')
<Arrow [2013-05-05T12:30:45+00:00]>
```

以下代码在字符串中搜索日期：

```
>>> arrow.get('June was born in May 1980', 'MMMM YYYY')
<Arrow [1980-05-01T00:00:00+00:00]>
```

arrow 时间模块库使用的字符串兼容 ISO-8601 标准：

```
>>> arrow.get('2013-09-30T15:34:00.000-07:00')
<Arrow [2013-09-30T15:34:00-07:00]>
```

arrow 对象也可以直接实例化，参数与 **datetime** 模块库的参数相同：

```
>>> arrow.get(2013, 5, 5)
<Arrow [2013-05-05T00:00:00+00:00]>

>>> arrow.Arrow(2013, 5, 5)
<Arrow [2013-05-05T00:00:00+00:00]>
```

6.5.4 arrow 属性

以下代码得到一个日期或时间戳：

```
>>> a = arrow.utcnow()
>>> a.datetime
datetime.datetime(2013, 5, 7, 4, 38, 15, 447644, tzinfo=tzutc())

>>> a.timestamp
1367901495
```

以下代码提取正常日期（如非夏日制等）和时区信息（tzinfo）：

```
>>> a.naive
datetime.datetime(2013, 5, 7, 4, 38, 15, 447644)
>>> a.tzinfo
tzutc()
```

以下代码提取时间分解数据：

```
>>> a.year
2013
```

以下代码调用 datetime 函数返回的属性：

```
>>> a.date()
datetime.date(2013, 5, 7)

>>> a.time()
datetime.time(4, 38, 15, 447644)
```

6.5.5 replace 替换和 shift 位移

请看以下代码：

```
>>> arw = arrow.utcnow()
>>> arw
<Arrow [2013-05-12T03:29:35.334214+00:00]>

>>> arw.replace(hour=4, minute=40)
<Arrow [2013-05-12T04:40:35.334214+00:00]>
```

可以向前或向后移动来修改时间，代码如下：

```
>>> arw.replace(weeks=+3)
<Arrow [2013-06-02T03:29:35.334214+00:00]>
```

6.5.6 format 格式化参数

使用 format 格式化参数的代码如下：

```
>>> arrow.utcnow().format('YYYY-MM-DD HH:mm:ss ZZ')
'2013-05-07 05:23:16 -00:00'
```


6.5.7 时间转换

可以通过名称或 `tzinfo` 时区属性转换数据，代码如下：

```
>>> utc = arrow.utcnow()
>>> utc
<Arrow [2013-05-07T05:24:11.823627+00:00]>

>>> utc.to('US/Pacific')
<Arrow [2013-05-06T22:24:11.823627-07:00]>

>>> utc.to(tz.gettz('US/Pacific'))
<Arrow [2013-05-06T22:24:11.823627-07:00]>
```

6.5.8 短命令

可以使用短命令，其代码如下：

```
>>> utc.to('local')
<Arrow [2013-05-06T22:24:11.823627-07:00]>

>>> utc.to('local').to('utc')
<Arrow [2013-05-07T05:24:11.823627+00:00]>
```

6.5.9 人性化

`arrow` 时间模块库提供人性化的输出信息，并且支持数十种语言：

```
>>> past = arrow.utcnow().replace(hours=-1)
>>> past.humanize()
'an hour ago'
```

下面再看一个例子：

```
>>> present = arrow.utcnow()
>>> future = present.replace(hours=2)
>>> future.humanize(present)
'in 2 hours'
```

(“在 2 小时内”)

新版本支持越来越多的本地化输出信息设置，可参考语言支持模块 `locales.py`，以下是俄语格式的时间输出信息：

```
>>> future = arrow.utcnow().replace(hours=1)
>>> future.humanize(a, locale='ru')
'через 2 час(а,ов)'
```

6.5.10 范围和跨度

`arrow` 模块库可以获取任何单位的时间跨度，代码如下：

```
>>> arrow.utcnow().span('hour')
(<Arrow [2013-05-07T05:00:00+00:00]>, <Arrow
[2013-05-07T05:59:59.999999+00:00]>)
```

时间和日期的上限和下限，例如某个月第一天和最后一天的日期，通常都需要借助专业模块库进行处理，参见以下代码：

```
>>> arrow.utcnow().floor('hour')
<Arrow [2013-05-07T05:00:00+00:00]>

>>> arrow.utcnow().ceil('hour')
<Arrow [2013-05-07T05:59:59.999999+00:00]>
```

下面是时间跨度的程序案例：

```
>>> start = datetime(2013, 5, 5, 12, 30)
>>> end = datetime(2013, 5, 5, 17, 15)
>>> for r in arrow.Arrow.span_range('hour', start, end):
...     print r
...
(<Arrow [2013-05-05T12:00:00+00:00]>, <Arrow
[2013-05-05T12:59:59.999999+00:00]>)
(<Arrow [2013-05-05T13:00:00+00:00]>, <Arrow
[2013-05-05T13:59:59.999999+00:00]>)
(<Arrow [2013-05-05T14:00:00+00:00]>, <Arrow
[2013-05-05T14:59:59.999999+00:00]>)
(<Arrow [2013-05-05T15:00:00+00:00]>, <Arrow
[2013-05-05T15:59:59.999999+00:00]>)
```

```
(<Arrow [2013-05-05T16:00:00+00:00]>, <Arrow
[2013-05-05T16:59:59.999999+00:00]>)
```

或者只是重复一个时间范围：

```
>>> start = datetime(2013, 5, 5, 12, 30)
>>> end = datetime(2013, 5, 5, 17, 15)
>>> for r in arrow.Arrow.range('hour', start, end):
...     print repr(r)
...
<Arrow [2013-05-05T12:30:00+00:00]>
<Arrow [2013-05-05T13:30:00+00:00]>
<Arrow [2013-05-05T14:30:00+00:00]>
<Arrow [2013-05-05T15:30:00+00:00]>
<Arrow [2013-05-05T16:30:00+00:00]>
```

6.5.11 工厂模式

arrow 时间模块库提供工厂模式，以便于用户扩充 arrow 对象数据和内置函数，不过这些内容相对比较复杂，而且用得比较少，在此不做深入介绍。

6.5.12 Token 特殊字符

arrow 时间模块库提供了大量的特殊字符（Token），可以用于相关的函数和代码中，如表 6-1 所示。

表 6-1 Token 特殊字符的输出格式和示例

Token 特殊字符	输出格式	示 例
Year（年）	YYYY	2000, 2001, 2002 ... 2012, 2013
	YY	00, 01, 02 ... 12, 13
Month（月）	MMMM	January, February, March ...
	MMM	Jan, Feb, Mar ...
	MM	01, 02, 03 ... 11, 12
	M	1, 2, 3 ... 11, 12

续表

Token 特殊字符	输出格式	示 例
Day of Year（年的某一天）	DDDD	001, 002, 003 ... 364, 365
	DDD	1, 2, 3 ... 364, 365
Day of Month（月的某一天）	DD	01, 02, 03 ... 30, 31
	D	1, 2, 3 ... 30, 31
	Do	1st, 2nd, 3rd ... 30th, 31st
Day of Week（周的某一天）	dddd	Monday, Tuesday, Wednesday ...
	ddd	Mon, Tue, Wed ...
	d	1, 2, 3 ... 6, 7
Hour（小时）	HH	00, 01, 02 ... 23, 24
	H	0, 1, 2 ... 23, 24
	hh	01, 02, 03 ... 11, 12
	h	1, 2, 3 ... 11, 12
AM / PM（上午/下午）	A	AM, PM, am, pm
	a	am, pm
Minute（分钟）	mm	00, 01, 02 ... 58, 59
	m	0, 1, 2 ... 58, 59
Second（秒）	ss	00, 01, 02 ... 58, 59
	s	0, 1, 2 ... 58, 59
Sub-second（微妙）	S...	0, 02, 003, 000006, 123123123123...
Timezone（时区）	ZZZ	Asia/Baku, Europe/Warsaw, GMT ...
	ZZ	-07:00, -06:00 ... +06:00, +07:00
	Z	-0700, -0600 ... +0600, +0700
Timestamp（时间戳）	X	1381685817

7

第 7 章

网络足彩数据抓取

本章的标题是“网络足彩数据抓取”，标题当中之所以用“抓取”而不用“采集”，是因为：

- 虽然目前是大数据时代，各种金融数据接口 API 到处泛滥，但足彩赔率的实时数据和历史数据一直没有稳定的免费 API 数据接口；
- 虽然有部分收费接口，但价格昂贵，对于广大个人用户而言，完全无法接受；
- 所有赔率数据必须通过 Web 网页模式下载，分析 HTML 信息，提取赔率数据。

7.1 500彩票网站数据接口的优势

目前，我国正式的彩票网站有近百家，其中大部分是上市企业或者上市企业的关联公司创建的，这些网站都能够提供各种足彩的赔率数据。

综合考虑之后，笔者选择了 500 彩票网站作为本书的数据源，其原因是：

- 500 彩票网站是第一家在国外上市的彩票类公司，管理规范，实力雄厚，网站数据都是源自国际权威的体育数据供应商；
- 500 彩票网站赔率数据结构化好，便于后期分析处理；
- 最重要的一点，在 500 彩票网站中可以采集 2010 年至今的历史赔率数据，其他很多类似网站在采集历史数据方面有许多限制。

在本书中，笔者选择 500 彩票网站作为专业足彩网站的代表，如果没有特别说

明，所有的足彩赔率数据都是源自 500 彩票网站。

7.1.1 案例 7-1：抓取赔率数据网页

案例 7-1 的文件名是 `zc701.py`，介绍使用 Python 语言编程抓取 500 彩票网站足彩赔率数据，主要代码如下：

```
#---1#
xtim='2010-01-02';
us0='http://trade.500.com/jczq/?date='
uss=us0+xtim
fss='tmp/500_'+xtim+'_utf8.htm';print('f,',fss)
rx=zweb.web_get001(uss)
htm=rx.text;zt.f_add(fss,htm,True)
#
#---2#
fss='tmp/500_'+xtim+'_gbk.htm';print('f,',fss)
zt.f_add(fss,htm,True,cod='GBK')
#
#---3#
fss='tmp/500_'+xtim+'.htm';print('f,',fss)
zweb.web_get001txt(uss,ftg=fss)
#
#---4#
xtim=arrow.now().format('YYYY-MM-DD');
uss=us0+xtim
fss='tmp/500_'+xtim+'.htm';print('f,',fss)
zweb.web_get001txt(uss,ftg=fss)
```

案例 7-1 运行后，会在 `tmp` 目录下产生多个 500 开头的 HTML 网页文件，其中一个文件包含“2010-01-02”，另一个文件包含运行当天的日期，如图 7-1 和图 7-2 所示。

因为 500 彩票网站的网页文件采用 GBK 编码格式，而案例中保存的文件是 utf8 格式，所以用浏览器打开时会出现乱码，一般可以单击鼠标右键，在弹出的快捷菜单中选择“utf-8 编码格式”，就可以正常显示了。

序号	赛事	截止时间	主队 VS 客队	投注区	赔率	500指数
0001	德甲联赛	2010-01-02	拜仁慕尼黑 VS 汉堡	主胜	1.47	2.33
0002	德甲联赛	2010-01-02	拜仁慕尼黑 VS 汉堡	平手	1.47	2.33
0003	德甲联赛	2010-01-02	拜仁慕尼黑 VS 汉堡	客胜	1.47	2.33
0004	德甲联赛	2010-01-02	拜仁慕尼黑 VS 汉堡	主胜	1.47	2.33
0005	德甲联赛	2010-01-02	拜仁慕尼黑 VS 汉堡	平手	1.47	2.33
0006	德甲联赛	2010-01-02	拜仁慕尼黑 VS 汉堡	客胜	1.47	2.33
0007	德甲联赛	2010-01-02	拜仁慕尼黑 VS 汉堡	主胜	1.47	2.33
0008	德甲联赛	2010-01-02	拜仁慕尼黑 VS 汉堡	平手	1.47	2.33
0009	德甲联赛	2010-01-02	拜仁慕尼黑 VS 汉堡	客胜	1.47	2.33
0010	德甲联赛	2010-01-02	拜仁慕尼黑 VS 汉堡	主胜	1.47	2.33

图 7-1 历史足彩赔率数据（2010 年）

序号	赛事	截止时间	主队 VS 客队	投注区	赔率	500指数
0001	德甲联赛	2017-01-19	拜仁慕尼黑 VS 汉堡	主胜	1.47	2.33
0002	德甲联赛	2017-01-19	拜仁慕尼黑 VS 汉堡	平手	1.47	2.33
0003	德甲联赛	2017-01-19	拜仁慕尼黑 VS 汉堡	客胜	1.47	2.33
0004	德甲联赛	2017-01-19	拜仁慕尼黑 VS 汉堡	主胜	1.47	2.33
0005	德甲联赛	2017-01-19	拜仁慕尼黑 VS 汉堡	平手	1.47	2.33
0006	德甲联赛	2017-01-19	拜仁慕尼黑 VS 汉堡	客胜	1.47	2.33
0007	德甲联赛	2017-01-19	拜仁慕尼黑 VS 汉堡	主胜	1.47	2.33
0008	德甲联赛	2017-01-19	拜仁慕尼黑 VS 汉堡	平手	1.47	2.33
0009	德甲联赛	2017-01-19	拜仁慕尼黑 VS 汉堡	客胜	1.47	2.33
0010	德甲联赛	2017-01-19	拜仁慕尼黑 VS 汉堡	主胜	1.47	2.33

图 7-2 实时足彩赔率数据（2017 年）

注意图 7-1（历史）和图 7-2（实时）两幅足彩赔率数据截图的差别，细心的读者会发现，图 7-1（历史）的表格中没有赔率数据。

读者不要着急，我们看看 006 场次“热刺对彼特堡”栏目右侧的数据列，里面有三个汉字“析、亚、欧”，如图 7-3 所示。

数据	500指数
析	-
亚	-
欧	-

图 7-3 数据列

用鼠标单击 006 场次对应的三个汉字“析、亚、欧”，会分别打开三个网页。

- “析”字，表示分析网页，<http://odds.500.com/fenxi/shuju-278181.shtml>。
- “亚”字，表示亚洲赔率，<http://odds.500.com/fenxi/yazhi-278181.shtml>。
- “欧”字，表示欧洲赔率，<http://odds.500.com/fenxi/ouzhi-278181.shtml>。

注意以上三个链接.shtml 前的数字都是 278181，这不是巧合，这个数字被称为 gid，是英文 game id 的意思，表示每场比赛的 id 编号，这个 gid 编码数字在 500 彩票网站是唯一的，每个数字对应一场独立的比赛。

遗憾的是，这个 gid 数字并非是全球统一的，在我国的足彩网站中也各不相同。如果读者发现不同的网站的同一场足球比赛采用的 gid 编码数字是相同的，那么说明这些网站都是采用了相同的数据源，或者是一些关联企业、业务合作机构。

单击数据列三个汉字“析、亚、欧”，打开的网页分别如图 7-4～图 7-6 所示。

托特纳姆热刺 阿森纳 4:0

赛前数据统计(13个)

托特纳姆热刺 阿森纳

托特纳姆热刺 VS 阿森纳 比赛历史

托特纳姆热刺 阿森纳

托特纳姆热刺 阿森纳

图 7-4 分析网页

托特纳姆热刺 阿森纳 4:0

赛前数据统计(13个)

托特纳姆热刺 阿森纳

托特纳姆热刺 VS 阿森纳 比赛历史

托特纳姆热刺 阿森纳

托特纳姆热刺 阿森纳

图 7-5 亚洲赔率网页

托特纳姆热刺 阿森纳 4:0

赛前数据统计(13个)

托特纳姆热刺 阿森纳

托特纳姆热刺 VS 阿森纳 比赛历史

托特纳姆热刺 阿森纳

托特纳姆热刺 阿森纳

图 7-6 欧洲赔率网页

- 图 7-4 的分析页面有球队近期比赛情况和网站推荐等数据。
- 图 7-5 显示的是亚洲赔率，简称亚赔，采用的不是常见的胜、负、平模式，而是一套复杂的让球、受让球体系，让球数可以出现 1/4 到数球的差别，一般个人用户很少涉及。
- 图 7-6 显示的是欧洲赔率，简称欧赔，采用的是最常见的胜、负、平模式，欧赔也有让球模式，不过是另外的体系了。

在“析、亚、欧”三组网页数据的顶部，都有一个共同的导航条，如图 7-7 所示。

图 7-7 比赛数据导航条

单击图 7-7 中的比赛数据导航条,可以获得更多的比赛数据,如投注分析、让球指数、大小指数、比分指数、走势分析和技术统计等。这些数据的具体作用,请读者自己查看网站的相关说明。

不同彩票网站的比赛数据导航条,可能形式不同,不过大体都类似。需要注意的是,各大网站中的投注分析一般仅指该网站自身的投注数据,有很大的局限性。

曾经获得英国 MBA 商业创新大奖的必发指数及其网站,可以提供全球各个地区的投注整体实时统计数据,这些数据参考价值很大。我国因为网络屏蔽,无法直接访问,专业的足彩研究机构可采用 VPN 或者收费通道,获取必发的各种数据。

在案例 7-1 中,我们使用了 Top 量化软件工具箱 ztools_web.py 网络模块中的网页抓取函数 web_get001,来抓取 500 网站的足彩赔率网页数据,函数代码如下:

```
def web_get001(url):
    try:
        rx= requests.get(url,headers=zt_headers) #获得网页,headers
    except:
        rx=None
        return None
    finally:
        return rx
#
return rx
```

由以上代码可以看出,web_get001 网页抓取函数非常简单,就是直接调用网络模块 requests 抓取网页,增加了容错语句,防止运行时出现“卡死”状态。这种容错模式也是处理网络信息抓取、IO 数据采集标准的编程模式。

7.1.2 网页数据实战操作技巧

web_get001 网页抓取函数虽然好,但对于中文的支持不是很理想,特别是对各种中文编码不能自动识别,要按用户指定的编码格式自动转换。

web_get001txt 函数是 web_get001 函数的升级版,代码如下:

```
def web_get001txt(url, ucod='gb18030', ftg='', fcod='gbk'):
    htm, rx = '', web_get001(url)
    if rx != None:
        xcod = rx.apparent_encoding; #print(xcod, uss)
        rx.encoding = xcod #gb-18030
        #dss = rx.text
        htm = rx.text; #print(htm)
        if xcod.upper() == 'UTF-8':
            #print('@@u8a'); #print(htm)
            htm = htm.replace('&nbsp;', ' ')
            css = htm.encode("UTF-8", 'ignore').decode("UTF-8", 'ignore')
            css = css.replace(u'\xfffd ', u' ')
            css = css.replace(u'\xa0 ', u' ')
            htm = css.encode("GBK", 'ignore').decode("GBK", 'ignore')
        #
        if ftg != '': zt.f_add(ftg, htm, True, cod=fcod)
    #
    return htm
```

由以上代码可以看出，web_get001txt 函数做了以下几个方面的增强。

- 函数本身不直接抓取网页，而是调用 web_get001 函数抓取网页。
- 通过 rx.encoding 属性，默认设置编码格式为最新的 GB18030 编码格式，该格式兼容 GBK、GB2312 等多种编码格式。
- 如果 ftg 文件名不是空字符串，则函数可以直接把抓取的内容保存为文件，默认编码格式为 GB18030。
- 对于 UTF-8 格式的网页，会过滤部分特殊字符，避免后续处理时出错。
- 默认使用最新的 GB18030 大字符编码，比 GBK 更强。

UTF-8 对于中文的支持比较差，我国网站一般都是采用 GBK 编码，500 彩票网站也是如此，我国流行的论坛程序 Discuz、PHPWind 也是重点推广 GBK 版本，只是对于面向国外用户的网站，才建议使用 UTF-8 版本。

UTF-8 和 GB18030 基本是兼容的，在 Python 3 编程实战中，程序代码一般都是英文，变量名称不要使用中文。数据文件中如果没有中文，默认也是 UTF-8 编码格式；数据文件中如果有中文，除非数据源使用 UTF-8 标准或者必须强制使用 UTF-8 标准，否则一般建议采用 GBK 编码格式。

至于 web_get001txt 函数中的 rx.encoding 属性，默认设置为 GB18030 编码格式，

这是在实战中抓取中文网页总结的经验，如果采用其他的编码格式，往往会给后续的数据工作带来各种莫名其妙的小问题，读者也可以试一下其他的编码格式，具体体会一下其中的细节。

事实上，在案例 7-1 中，第一组代码中的历史数据如果不使用 2010-01-02，而是使用其他日期，则可能引发错误，例如：

```
xtim='2010-01-20';
```

系统运行出错，无法运行：

```
File "E:/00zbook/zc2016/zxc201wget.py", line 40, in <module>
    zt.f_add(fss,htm,True,cod='GBK')
```

```
File "E:\00zbook\zc2016\zTools.py", line 506, in f_add
    f.write(dss+'\n')
```

```
UnicodeEncodeError: 'gbk' codec can't encode character '\ufffd' in
position 113997: illegal multibyte sequence
```

计算机字符编码非常烦琐、复杂，有兴趣的读者可以搜索关键词 UTF-8、GBK、汉字编码、Python 内码等，查询更多的细节。

7.2 网页解析的心灵鸡汤

Beautiful Soup 的意思是“可口的汤羹”、“甲鱼汤”，是一套 Python 模块库，用于 HTML 网页的文本分析、数据清洗，目前已经发展到第 4 代。

Beautiful Soup 模块库使用起来非常方便，可以大幅节省用户的编程时间。作为 HTML 解析器，它可以很好地处理不规范标记并生成语法解析树（parse tree）。它提供简单又常用的导航（navigating）、搜索及修改语法解析树的操作。

简单地说，Beautiful Soup 是 Python 的一个模块库，最主要的功能是从网页中抓取数据，官方解释如下。

- Beautiful Soup 提供一些简单的、Python 式的函数用来处理导航、搜索、修改语法解析树等功能。它是一个工具箱，通过解析文档，为用户提供需要抓取的数据，因为简单，所以不需要多少代码就可以写出一个完整的应用程序，为程序员灵活地提供不同的解析策略或强劲的速度。
- Beautiful Soup 相比其他的 HTML 解析器有一个非常重要的优势，HTML 文档

会被拆解为对象处理，全篇转换为字典和数组。相比正则解析的爬虫工具，省略了学习正则的时间成本。

- BeautifulSoup 自动将输入文档转换为 Unicode 编码，将输出文档转换为 UTF-8 编码。用户不需要考虑编码方式，除非文档没有指定编码方式。
- BeautifulSoup 已成为和 lxml、html6lib 一样出色的 Python 解释器，为用户灵活地提供不同的解析策略或强劲的速度。

7.2.1 BS4 四大要素三缺一

Beautiful Soup 4 (BS4) 模块库将复杂的 HTML 文档转换成一个复杂的树形结构，每个节点都是 Python 对象，所有对象可以归纳为以下 4 种。

- Tag，标签对象。
- Navigable String，导航字符。
- BeautifulSoup4，BS4 复合对象。
- Comment，注释。

虽然 BeautifulSoup 4 模块库包括以上四大基本要素，但 Comment（注释）对象实际上在分析 HTML 网页时，特别是提取数据时使用得很少，所以下面重点介绍前面三种对象。因此说：四大要素三缺一。

7.2.2 Tag 标签对象

Tag 标签对象是 HTML 网页格式的一个术语，相当于一个网络节点的 ID 标识符，BS4 模块库中的 Tag 标签对象，对应的正是 HTML 中的一个标签，具体使用方法读者可以参看案例 7-2。

7.2.3 案例 7-2：Tag 标签对象

案例 7-2 的文件名是 zc702_bs4.py，介绍了 BeautifulSoup 4 模块库的一些最基本、最常用的功能，主要代码如下：

```

def bs010(fsr):
    hss=zt.f_rd(fsr);
    bs=BeautifulSoup(hss,'html5lib') # 'lxml'
    #bs=BeautifulSoup(hss,'lxml') # 'lxml'
    #
    print('bs.title.name,',bs.title.name)
    print('bs.title.string,',bs.title.string)
    print('bs.title,',bs.title)
    print('\n')
    print('\n@bs.p\n',bs.p)
    print('\n@bs.a\n',bs.a)
    print('\n@bs.find(id="p_text")\n',bs.find(id='p_text'))

#-----
fss='dat/500_2017-01-20.htm';print('f,',fss)
bs010(fss)

```

案例 7-2 运行输出的信息如下:

```

f, dat/500_2017-01-20.htm
bs.title.name, title
bs.title.string, 170120 竞彩足球投注-500 彩票网
bs.title, <title>170120 竞彩足球投注-500 彩票网</title>

@bs.p
<p class="p_tit">短信获取下载地址</p>

@bs.a
<a class="tips_hd" data_tongji="top_sj"
href="http://www.500.com/wap/" rel="nofollow" target="_blank"><span
class="topbar_icon icon_phone"></span>手机版<s class="arrow"></s></a>

@bs.find(id="p_text")
<input class="p_text" id="p_text" maxlength="11" type="text" value="
您的手机号"/>

```

案例 7-2 使用的数据文件名是 dat/500_2017-01-20.htm, 这个数据文件是笔者测试时抓取的 500 彩票网站实时足彩赔率数据, 为了保持用户体验统一, 防止用户删除, 我们把该文件复制到 dat 数据目录下。

案例 7-2 输出的 title 数据对应的网页 HTML 源码如图 7-8 所示。



```

500_2017-01-20.htm x
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1"
2 <html xmlns="http://www.w3.org/1999/xhtml">
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
5 <title>170120竞彩足球投注-500彩票网</title>
6 <meta name="Keywords" content="170120竞彩足球投注" />
7 <meta name="Description" content="500彩票网竞彩足球购彩大厅提供170120竞彩足球合买代购、
8 <link href="http://www.500cache.com/trade/css/public.css?v=2015030201" type="text/css"
9 <link href="http://www.500cache.com/trade/css/zc_public.css?v=2015012601" type="text/css"
10 <link href="http://www.500cache.com/trade/css/task.css?v=2014072401" type="text/css"
11 <link rel="canonical" href="http://trade.500.com/jczq/" />
12 <style>
13 .l_btn_agent, .l_btn_hm, .btn_agent, .btn_hm{zoom:1;}

```

图 7-8 HTML 源码包含的 title 数据

需要说明的是，不少网站经常修改网页源码，因此本书的内容、截图如果与实际网站的截图、信息有所差异，属于正常现象。

在案例 7-2 中：

- bs 就是 BeautifulSoup 处理格式化后的字符串；
- bs.title 得到的是 title 标签的数据；
- bs.p 得到的是文档中的第一个 p 标签，要想得到所有标签，需要用 find_all 函数。

在案例 7-2 中，利用 BeautifulSoup 模块库，类似对象属性，加上标签名就可以轻松地获取相关标签的内容，比使用正则表达式还要方便。

不过在案例 7-2 中，查找的是在所有内容中的第一个符合要求的标签，如果要查询所有的标签，需要使用 find_all 等函数。

因为 find_all、get_text 函数输出太多，所示案例 7-2 中没有使用，读者可以自己试一下。

- find_all 函数返回的是一个序列，可以对它进行循环，依次得到想要的东西。
- get_text 函数返回的是文本，对每一个 BeautifulSoup 处理后的对象得到的标签都是生效的，可以试试 print(bs.p.get_text())。

Beautiful Soup 4 模块库的其他常用功能如下。

- 获得标签的其他属性，比如要获得 a 标签的 href 属性的值，可以使用 bs.a['href']，类似的其他属性如 class，也可以使用 bs.a['class']。
- 一些特殊的标签，比如 title、head 标签，可以通过 bs.title、bs.head 得到，其实前面也已经说了。

- 使用 `contents` 属性可以获得标签的内容数组，比如使用 `bsp.head.contents` 就获得了 `head` 下的所有子节点，以列表的形式返回结果。
- 内容数组可以使用 `[num]` 形式获得，再使用 `.name` 就可以获得标签。
- 获取标签的节点也可以使用 `children`，但是不能直接使用 `bs.head.children`，否则不会返回列表，返回的是 `listiterator` 对象，需要使用 `list` 将其转换为列表，或者使用 `for` 语句遍历提取相关的数据。
- `string` 属性中若超过一个标签，那么就会返回 `None`，否则就返回具体的字符串。案例 7-2 中 `print(bs.title.string)` 就返回了“【竞彩足球】投注|合买|代购_中国竞彩网首页_500 彩票网”。
- 超过一个标签，可以使用 `strings`。
- 向上查找可以用 `parent` 函数，如果查找所有的节点，那么可以使用 `parents` 函数。
- 查找下一个兄弟节点使用 `next_sibling`，查找上一个兄弟节点使用 `previous_sibling`，如果是查找所有的节点，那么在对应的函数后面加 `s` 就可以了。

在使用 `Beautiful Soup` 对象时，要明确解析器 `bs = BeautifulSoup(html, 'html5lib')`，否则写成 `soup = BeautifulSoup(html)` 会有警告信息。

在案例 7-2 中，底层使用的 HTML 解析库是 `html5lib`，而不是默认的 `lxml`，相关代码如下：

```
bs=BeautifulSoup(hss,'html5lib') # 'lxml'
#bs=BeautifulSoup(hss,'lxml') # 'lxml'
```

这其中的原因很多，主要有：

- 理论上 `lxml` 速度相对快一点，许多强调速度的程序可以使用 `lxml` 模块库；
- `lxml` 不是纯 Python 模块库，需要下载二进制安装包；
- `lxml` 解析器对中文标签支持不好，会缺失很多内容，特别是 UTF-8 编码格式，需要转换为 Unicode 格式；
- `html5lib` 采用了部分 HTML5 标准，容错性更好，可以自动补全部分缺失的属性标签。

7.2.4 案例 7-3：Tag 标签对象数据类型

案例 7-3 的文件名是 `zc703_bs4_type.py`，用于进一步分析 Tag 标签对象的数据

类型。开头语句与案例 7-2 相同，都是读取赔率数据文件 dat/500_2017-01-20.htm，并转换为 BS 对象：

```
hss=zt.f_rd(fsr);
bs=BeautifulSoup(hss,'html5lib') # 'lxml'
```

案例 7-3 代码虽然简单，但是毕竟较长，我们将分组进行讲解。

第 1 组语句用于查看 BS4 对象的基本类型：

```
print('\n@ bs.type')
print('\ntype:bs,',type(bs))
print('\ntype:bs.title,',type(bs.title),'\n',bs.title)
print('\ntype:bs.title.name,',type(bs.title.name),'\n',bs.title.name)
print('\ntype:bs.title.attrs,',type(bs.title.attrs),'\n',bs.title.attrs)
print('\ntype:bs.title.string,',type(bs.title.string),'\n',bs.title.string)
print('\ntype:bs.title.strings,',type(bs.title.strings),'\n',bs.title.strings)
```

笔者特意在程序中加入了大量的“\n”换行符号，便于分组输出数据，以上代码对应的输出信息如下：

```
@ bs.type #1

type:bs, <class 'bs4.BeautifulSoup'>

type:bs.title, <class 'bs4.element.Tag'>
<title>170120 竞彩足球投注-500 彩票网</title>

type:bs.title.name, <class 'str'>
title

type:bs.title.attrs, <class 'dict'>
{}

type:bs.title.string, <class 'bs4.element.NavigableString'>
170120 竞彩足球投注-500 彩票网

type:bs.title.strings, <class 'generator'>
<generator object _all_strings at 0x000002AE4F761518>
```


由输出信息可以看出：

- BS 对象本身还是一个 BeautifulSoup 对象，类似递归定义；
- bs.title 是一个 bs4.element.Tag 标准的 Tag 标签对象；
- bs.title.name 是 title 对象名称，是 str 字符串对象；
- bs.title.attrs 是 title 对象属性参数，字典格式，没有数据，返回空字典对象；
- bs.title.string 是 bs4.element.NavigableString 导航字符串对象；
- bs.title.strings 是 string 属性的复数形式，如果存在多个对象，则返回的对象是列表格式。

对于 Tag 标签对象而言，它有两个重要的基本参数，即 name（名称）和 attrs（属性）。

在案例 7-3 中，对应的脚本程序第 2 组代码如下：

```
print('\n\n@ bs.type #2')
print('\ntype:bs.a, ', type(bs.a), '\n', bs.a)
print('\ntype:bs.a.name, ', type(bs.a.name), '\n', bs.a.name)
print('\ntype:bs.a.attrs, ', type(bs.a.attrs), '\n', bs.a.attrs)
```

第 2 组代码对应的输出信息是：

```
@ bs.type #2

type:bs.a, <class 'bs4.element.Tag'>
<a class="tips_hd" data_tongji="top_sj"
href="http://www.500.com/wap/" rel="nofollow" target="_blank"><span
class="topbar_icon icon_phone"></span>手机版<s class="arrow"></s></a>

type:bs.a.name, <class 'str'>
a

type:bs.a.attrs, <class 'dict'>
{'target': '_blank', 'rel': ['nofollow'], 'data_tongji': 'top_sj',
'href': 'http://www.500.com/wap/', 'class': ['tips_hd']}
```

第 2 组代码对应输出 bs.a 标签对象的参数，其中类型、名称、包含的代码都大同小异。需要注意的是，bs.a.attrs 属性参数的输出是一个字典对象：

```
{'target': '_blank', 'rel': ['nofollow'], 'data_tongji': 'top_sj',
'href': 'http://www.500.com/wap/', 'class': ['tips_hd']}
```

bs.a.attrs 属性参数返回字典对象数据，对应的 HTML 源码保存在 bs.a 标签对象当中：

```
<a class="tips_hd" data_tongji="top_sj" href="http://www.500.com/wap/"
rel="nofollow" target="_blank"><span class="topbar_icon
icon_phone"></span>手机版<s class="arrow"></s></a>
```

请读者将 `bs.a.attrs` 属性参数返回的字典数据与 `bs.a` 标签对象中的 HTML 源码进行逐一对应分析。

第 3 组代码进一步说明 `bs.a.attrs` 属性参数的使用，通过字典格式调用相关的数据，相关代码如下：

```
print('\n\n@ bs #3')
print('\ntype:bs.a["class"],',type(bs.a['class']),bs.a['class'])
print('\nbs.a["rel"],',bs.a['rel'])
print('bs.a["target"],',bs.a['target'])
print('bs.a["href"],',bs.a['href'])
print('bs.a["data_tongji"],',bs.a['data_tongji'])
```

第 3 组代码对应的输出信息如下：

```
@ bs #3

type:bs.a["class"], <class 'list'> ['tips_hd']

bs.a["rel"], ['nofollow']
bs.a["target"], _blank
bs.a["href"], http://www.500.com/wap/
bs.a["data_tongji"], top_sj
```

第 3 组代码输出的信息就是标准的字典对象数据，需要注意的是其中的 `type` 语句：

```
print('\ntype:bs.a["class"],',type(bs.a['class']),bs.a['class'])
```

返回的是 `list` 对象格式，而不是字符串：

```
type:bs.a["class"], <class 'list'> ['tips_hd']
```

这说明部分返回的数据是列表格式，当然也有部分返回的数据是字符串格式，读者使用时要注意细节。

Python 的字典数据格式可以使用 `get` 属性函数，提取保存的数值，BS4 的 Tag 标签对象也可使用 `get` 属性函数，案例中第 4 组代码如下：

```
print('\n\n@ bs #4')
print('bs.a.get("class"),',bs.a.get('class'))
print('bs.a.get("rel"),',bs.a.get('rel'))
print('bs.a.get("target"),',bs.a.get('target'))
```

```
print('bs.a.get("href"),',bs.a.get('href'))
print('bs.a.get("data_tongji"),',bs.a.get('data_tongji'))
```

第4组代码的输出数据与第3组代码的输出数据一样，在此不再重复。

此外，我们可以像修改、删除字典数据一样，修改、删除 BS4 标签对象的相关属性参数，有兴趣的读者可以自己测试一下。

7.2.5 NavigableString 导航字符串

在案例 7-3 中，第 1 组代码其中的两行程序是：

```
print('\ntype:bs.title,',type(bs.title),'\n',bs.title)
print('\ntype:bs.title.string,',type(bs.title.string),'\n',bs.title.s
tring)
```

对应的输出信息是：

```
type:bs.title, <class 'bs4.element.Tag'>
<title>【竞彩足球】投注|合买|代购_中国竞彩网首页_500 彩票网</title>

type:bs.title.string, <class 'bs4.element.NavigableString'>
【竞彩足球】投注|合买|代购_中国竞彩网首页_500 彩票网
```

其中，bs.title.string 中的.string 属性参数，其数据类型就是 NavigableString 导航字符串，其内容就是标签之间的文字。

7.2.6 BeautifulSoup 复合对象

在案例 7-3 中，第 1 组代码中的以下脚本用于检查 BS 对象的数据类型：

```
print('\ntype:bs,',type(bs))
```

对应的输出信息是：

```
type:bs, <class 'bs4.BeautifulSoup'>
```

数据类型也是 BeautifulSoup 格式，类似递归定义，在实际分析时，大多时候可以把 BS 对象当做一个特殊的 Tag 标签对象，分别获取它的类型、名称、属性等参数。

在案例 7-3 中，第 5 组代码就是演示 BS 对象的相关参数，程序代码如下：

```
print('\n\n@ bs #5')
print('type:bs.name,',type(bs.name))
```

```
print('bs.name,',bs.name)
print('bs.attrs,',bs.attrs)
print('bs.string,',bs.string)
print('bs.strings,',bs.strings)
```

对应的输出信息如下：

```
@ bs #5
type:bs.name, <class 'str'>
bs.name, [document]
bs.attrs, {}
bs.string, None
bs.strings, <generator object _all_strings at 0x0000016970BDCDB0>
```

需要说明的是，案例 7-3 中 `bs.name` 的数据类型在 Python 3 以上版本中默认为字符串类型。如果是 Python 2 版本，则类型是 Unicode 长字符串。如果采用了其他编码格式，则输出信息可能会有其他稍许变化。

`bs.name` 的数据值是 `[document]`，这可能源自 HTML 的 DOM 结构，根节点就是 `document`；当读者向上检索父节点，发现节点名称为 `name`、数值为 `document` 时，就表示到了根节点。

7.2.7 Comment 注释对象

Comment 注释对象本质上是一种特殊类型的 `NavigableString` 导航字符串对象，如果处理不好，可能会给文本处理带来意想不到的麻烦。因为 Comment 注释对象使用得较少，所示具体细节请读者参看相关文档。

7.2.8 案例 7-4：BS4 查找匹配功能

案例 7-4 的文件名是 `zc704_bs4_find.py`，主要介绍在 Beautiful Soup 4 模块库中使用 `select`、`find` 和 `find_all` 等函数来配合标签、属性等多种参数，匹配、提取相关的数据。

案例 7-4 首先说明 `select` 函数的使用，有关代码如下：

```
#---select
```

```
print("\n@bs.select('title',limit=5):\n",bs.select('title',limit=5))
print("\n@bs.select('a',limit=5):\n",bs.select('a',limit=5))
print("\n@bs.select('#p_text',limit=5):\n",bs.select('#p_text',limit=
5))
```

对应的输出信息如下:

```
@bs.select('title',limit=5):
[<title>170120 竞彩足球投注-500 彩票网</title>]

@bs.select('a',limit=5):
[<a class="tips_hd" data_tongji="top_sj"
href="http://www.500.com/wap/" rel="nofollow" target="_blank"><span
class="topbar_icon icon_phone"></span>手机版<s class="arrow"></s></a>, <a
class="btn_orange btn_orange_h24 btn_w70" data_tongji="top_dx"
href="javascript:void(0)">免费发送</a>, <a class="blue"
href="javascript:void(0)">再次发送短信</a>, <a class="topbar_login"
data_tongji="top_dl" href="javascript: void(0)" id="top_login_btn">请登录
</a>, <a class="topbar_reg" data_tongji="top_zc"
href="http://passport.500.com/user/" rel="nofollow" target="_blank">免费注
册</a>]

@bs.select('#p_text',limit=5):
[<input class="p_text" id="p_text" maxlength="11" type="text" value="
您的手机号"/>]
```

`select` 函数用于选择 CSS 标签,是 Beautiful Soup 4 新增加的功能,类似 `find_all`,不过在节点提取方面增强了很多。标签名称前面的“#”表示查找对应的 id;“~”表示所有其他兄弟标签;“+”表示第一个其他兄弟标签。

`select` 函数和 `find_all` 函数返回的都是多个匹配数据,以列表形式返回,案例中增加了参数 `limit=5`,限制最多输出前 5 个匹配的数据。

案例 7-4 中 `find_all` 函数的有关代码如下:

```
#---find all
print("\n@bs.find_all('a',limit=5):\n",bs.find_all('a',limit=5))
print("\n@bs.find_all('p',limit=5):\n",bs.find_all('p',limit=5))
```

对应的输出信息是:

```
@bs.select('#p_text',limit=5):
[<input class="p_text" id="p_text" maxlength="11" type="text" value="
```

您的手机号"/>]

```
@bs.find_all('a',limit=5):
    [<a class="tips_hd" data_tongji="top_sj"
href="http://www.500.com/wap/" rel="nofollow" target="_blank"><span
class="topbar_icon icon_phone"></span>手机版<s class="arrow"></s></a>, <a
class="btn_orange btn_orange_h24 btn_w70" data_tongji="top_dx"
href="javascript:void(0)">免费发送</a>, <a class="blue"
href="javascript:void(0)">再次发送短信</a>, <a class="topbar_login"
data_tongji="top_dl" href="javascript: void(0)" id="top_login_btn">请登录
</a>, <a class="topbar_reg" data_tongji="top_zc"
href="http://passport.500.com/user/" rel="nofollow" target="_blank">免费注册
</a>]
```

可以看到，虽然用 `limit` 参数做了限制，但输出数据还是不少。`find_all` 函数通常用于对新的数据文档的初步提取，列出有关数据，为进一步精确匹配提供标签名称。

在案例 7-4 中，精确匹配提取也是使用 `find_all` 函数，不过增加了属性参数，有关代码如下：

```
#---find_all+type
print("\n@bs.find_all('a', class_='topbar_reg'):\n",bs.find_all('a',
class_='topbar_reg'))
print("\n@bs.find_all('a', rel='nofollow',limit=6):\n",bs.find_all
('a', rel='nofollow',limit=5))
print("\n@bs.find_all('a', attrs={'class':
'topbar_reg'}):\n",bs.find_all('a', attrs={'class': 'topbar_reg'}))
#
```

对应的输出信息是：

```
@bs.find_all('a', class_='topbar_reg'):
    [<a class="topbar_reg" data_tongji="top_zc"
href="http://passport.500.com/user/" rel="nofollow" target="_blank">免费注册
</a>]

@bs.find_all('a', rel='nofollow',limit=3):
    [<a class="tips_hd" data_tongji="top_sj" href="http://www.500.com/
wap/" rel="nofollow" target="_blank"><span class="topbar_icon
icon_phone"></span>手机版<s class="arrow"></s></a>, <a class="topbar_reg"
```

```
data_tongji="top_zc" href="http://passport.500.com/user/" rel="nofollow"
target="_blank">免费注册</a>, <a class="tips_hd" data_tongji="top_sj"
href="http://www.500.com/wap/" rel="nofollow" target="_blank"><span
class="topbar_icon icon_phone"></span>手机版<s class="arrow"></s></a>, <a
class="tips_hd" data_tongji="top_yhm" href="http://trade.500.com/
useraccount/?254439398" rel="nofollow" target="_blank"></a>, <a
class="vipdengji_con" href="http://vip.500.com" rel="nofollow"
target="_blank" title="前往会员中心"><span class="vipdengji_v" id="hydj_p"
style="width: 40%;"></span><span class="vipdengji_text"
id="hydj"></span></a>]
```

```
@bs.find_all('a', attrs={'class': 'topbar_reg'}):
    [<a class="topbar_reg" data_tongji="top_zc" href="http://
passport.500.com/user/" rel="nofollow" target="_blank">免费注册</a>]
```

案例 7-4 中的精确匹配程序使用了几种不同的方式匹配属性, 程序第 1 句和第 3 句的输出信息都是一条, 说明已经是精确匹配; 程序第 2 句的输出是一个列表, 包括多个匹配数据, 如果不是用户需要, 就说明需要使用其他的匹配参数, 提高匹配精度。当然, 如果找不到匹配对象, 返回值是空列表。

案例 7-4 最后演示了根据文本查找匹配数据, 程序脚本如下:

```
#---find_all+text
print("\n@bs.find_all(text='手机版'):\n",bs.find_all(text='手机版'))
```

输出信息是:

```
@bs.find_all(text='手机版'):
    ['手机版', '手机版']
```

需要说明的是, 案例当中的文本匹配没有使用正则表达式, 所以必须是完全匹配。

BS4 模块库函数中很多参数都支持正则表达式, 这大大增强了软件的灵活性, 限于篇幅, 这方面的资料请读者自己浏览官方网站。

BS4 模块库函数也支持列表形式的多组参数, 如:

```
Bs.find_all(['手机','笔记本','台式机'])
```

这种模式类似传统的“or”或运算, BS4 会将与列表中任意元素匹配的内容返回。

BS4 模块库函数传入的参数也可以是函数, 这个将在后面的案例中具体说明。

7.2.9 BS4 节点遍历功能

BS4 模块库还支持节点遍历功能，包括子节点、父节点、兄弟节点等多种形式，这方面的主要函数和属性如下。

- `.contents` 属性，可以将 Tag 的子节点以列表的方式输出，可以用列表索引来获取它的某一个元素。
- `.children` 属性，是一个 list 生成器对象，可以遍历获取相关数据。
- `.descendants` 属性，所有的子孙节点。`.contents` 和 `.children` 属性仅包含 Tag 对象的直接子节点，`.descendants` 属性可以对所有 Tag 的子孙节点进行递归循环，和 `.children` 类似，也需要遍历获取其中的内容。
- `.string` 属性，节点内容。如果 Tag 对象只有一个 `NavigableString` 类型子节点，那么这个 Tag 可以使用 `.string` 得到子节点。如果一个标签里面没有其他标签了，那么 `.string` 就会返回标签里面的内容。如果标签里面只有唯一的一个标签，那么 `.string` 会返回最里面的内容。
- `.strings`，获取多个内容，不过需要遍历获取。
- `.stripped_strings`，输出的字符串中可能包含了很多空格或空行，使用该属性可以去除多余空白内容。
- `.parent` 属性，父节点，即上一级对象的节点。
- `.parents` 属性，全部父节点，通过元素的 `.parents` 属性可以递归得到元素的所有父节点。
- `.next_sibling` 和 `.previous_sibling` 属性，兄弟节点。兄弟节点可以理解为和本节点处在同一级的节点，`.next_sibling` 属性获取了该节点的下一个兄弟节点，`.previous_sibling` 属性则与之相反，如果节点不存在，则返回 `None`。实际文档中 Tag 的 `.next_sibling` 和 `.previous_sibling` 属性通常是字符串或空白，因为空白或者换行也可以被视做一个节点，所以得到的结果可能是空白或者换行。
- `.next_siblings` 和 `.previous_siblings` 属性，全部兄弟节点，通过 `.next_siblings` 和 `.previous_siblings` 属性可以对当前节点的兄弟节点迭代输出。
- `.next_element` 和 `.previous_element` 属性，前后节点，与 `.next_sibling` 和 `.previous_sibling` 不同，它并不是针对兄弟节点，而是针对所有节点，不分层次。

- `.next_elements` 和 `.previous_elements` 属性，所有前后节点。通过 `.next_elements` 和 `.previous_elements` 的迭代器就可以向前或向后访问文档的解析内容，就好像文档正在被解析一样。

节点遍历需要了解 HTML 网页文件的 DOM 模型，这方面的细节请读者自行参考有关的资料。

7.3 足彩基本数据抓取

前面已经讲解了抓取 500 彩票网站的赔率数据网页的方法，下面通过具体案例说明如何分析抓取后的 500 彩票网站的历史足彩赔率网页和实时足彩赔率网页，以及如何通过专业的 BeautifulSoup 等 HTML 网页分析模块，从中提取对应的网页数据。

7.3.1 案例 7-5：分析网页比赛数据

前面讲解了根据制定的日期，从 500 彩票网站抓取比赛数据网页，也介绍了 gid 比赛基本数据文件的数据构成，下面具体分析 500 彩票网站的比赛信息网页，学习如何从中提取 gid 比赛数据文件所需的信息。

案例 7-5 的文件名是 `zc705_hmchk01.py`，说明如何分析和提取网页的基本比赛数据，主要代码如下：

```
def htm_chk001(htm):
    bs=BeautifulSoup(htm,'html5lib') # 'lxml'

    #---1#
    zsys.bs_get_ktag_kstr='isend'
    x10=bs.find_all(zweb.bs_get_ktag)
    for xc,x in enumerate(x10):
        print('\n@x\n',xc,'#',x.attrs)

    #-----
    fss='dat/500_2017-01-20.htm';print('f,',fss)
    hss=zt.f_rd(fss,cod='gbk')
    htm_chk001(hss)
```

案例 7-5 运行结果很长，下面只截取了部分输出信息：

```
@x
0 # {'fid': '581438', 'rq': '-1', 'awaysxname': '阿德莱', 'gdate': '星
期五 201', 'win': '0', 'dgactive': '{"spf":"0","jq":"1","bf":"1","bq":
"1","nspf":"0"}', 'lost': '0', 'lg': '澳超', 'draw': '0', 'pdate':
'2017-01-20', 'pendtime': '2017-01-20 16:45:00', 'zid': '106506', 'class':
['even', ''], 'isend': '1', 'homesxname': '悉尼 FC', 'pname': '5001', 'mid':
'120625'}

@x
1 # {'fid': '632530', 'rq': '-1', 'awaysxname': '斯托曼', 'gdate': '星
期五 201', 'win': '0', 'dgactive': '{"spf":"0","jq":"1","bf":"1","bq":
"1","nspf":"0"}', 'lost': '0', 'lg': '球会友谊', 'draw': '0', 'pdate':
'2017-01-20', 'pendtime': '2017-01-20 19:25:00', 'zid': '106536', 'class':
['odd', ''], 'isend': '1', 'homesxname': '瓦勒伦', 'pname': '5031', 'mid':
'120655'}
```

可以看出，输出信息已经是基本结构化的信息了，可以很容易地转换为 Pandas（潘达思）的 DataFrame 数据格式。

案例 7-5 的程序很简单，主要是调用 htm_chk001 函数分析网页信息。

在 htm_chk001 函数代码中，x10 节点数据的提取代码只有两行：

```
zsys.bs_get_ktag_kstr='isend'
x10=bs.find_all(zweb.bs_get_ktag)
```

代码中 zsys.bs_get_ktag_kstr 是一个全局变量，用于设置 zt.bs_get_ktag 函数中的特征字符串。

bs_get_ktag_kstr 全局变量也可以与 bs_get_ktag 放在同一个 ztools_web.py 模块，不过使用时需要加 global 参数，还要提前声明，否则很容易因为变量的作用范围出现错误，有兴趣的读者可以自己试一下。

Beautiful Soup 的许多函数，例如 find_all，不仅支持变量数据，也支持函数作为输入参数，这个也是 Python 语言灵活多变、功能强大的一个体现。传统的 C 语言、Delphi 要实现类似的功能非常复杂。

bs_get_ktag 函数位于 ztools_web.py 模块，用于提取符合要求的特征码，代码如下：

```
def bs_get_ktag(tag):
    #return tag.has_attr('isend')
```

```
#print('k',fb_get_ktag_kstr)
return tag.has_attr(zsys.bs_get_ktag_kstr)
```

bs_get_ktag 函数的操作技巧在 Python 语言中很常见,在使用 Matplotlib 绘制无空隙 K 线图时,也使用了类似的编程技巧。

至于为什么采用字符串“isend”,这个主要是用于分析网页源码,网页源码截图如图 7-9 所示,根据经验,多测试几次,一般就可以找到所需要的字符串作为特征码。

```
778 </colgroup>
779 <tr zid="106506" mid="120625" pname="5001" pdate="2017-01-20" lg="澳超" rq="-1" pendtime="2017-01-2
- 0 16:45:00" win="0" draw="0" lost="0" isend="1" gdate="星期五 201" fid="581438" homesxname="悉尼FC
- " awaysxname="阿德莱" class="even" dactive="('spf':"0","jq":"1","bf":"1","bq":"1","nsfp":"0")">
780 <td ><a href="javascript: void(0)" class="game_num">001<s></s></a></td>
781 <td><span class="league" style="background: #336699"><a href="http://liansai.500.com/zuqiu-3853/" t
- target="_blank">澳超</a></td>
782 <td>
783 <span class="match_time" title="开赛时间: 2017-01-20 16:50" style="display:">16:50</span>
784 <span class="end_time" title="截止时间: 2017-01-20 16:45:00" style="display:none">16:45<br /><!--<span
- class="gray">暴雨</span>--></span>
785 <span class="countdown_time" title="剩余时间" style="display:none" data-endtime="2017-01-20 16:45:00">1
- 6:45</span>
786 </td>
```

图 7-9 网页源码截图

提取了节点 x10 的数据后,打印输出 x10 节点中的数据和结构,这是一种常规的网页信息编程技巧,案例 7-5 中的代码是:

```
for xc,x in enumerate(x10):
    print('\n@x\n',xc,'#',x.attrs)
```

7.3.2 案例 7-6: 提取网页比赛数据

前面的案例已经把主要的比赛信息转换为结构化数据,有了转换后的结构化数据,再提取 gid 比赛数据基本上就是水到渠成的事情。

案例 7-6 的文件名是 zc706_gidget01.py,说明如何提取 gid 数据,核心代码如下:

```
def gid_get001(htm):
    bs=BeautifulSoup(htm,'html5lib') # 'lxml'
    df=pd.DataFrame(columns=tfsys.gidSgn,dtype=str)
    ds=pd.Series(tfsys.gidNil,index=tfsys.gidSgn,dtype=str)

    #---1#
    zsys.bs_get_ktag_kstr='isend'
    x10=bs.find_all(zweb.bs_get_ktag)
    for xc,x in enumerate(x10):
```

```

print('\n@x\n',xc,'#',x.attrs)
#print('\n@x\n',xc,'#',x.attrs)
ds['gid'],ds['gset']=x['fid'],zstr.str_fltrHtmHdr(x['lg'])
ds['mplay']=zstr.str_fltrHtmHdr(x['homesxname'])
ds['gplay']=zstr.str_fltrHtmHdr(x['awaysxname'])
ds['kend']=x['isend']
ds['tweek']=x['gdate'].split(' ')[0] #tweek
ds['tplay'],ds['tsell']=x['pdate'],x['pendtime']
#tplay,tsell,
#
df=df.append(ds.T,ignore_index=True)

#---
df=df[df['gid']!='-1']
return df

#-----
fss='dat/500_2017-01-20.htm';print('f',fss)
hss=zt.f_rd(fss,cod='gbk')
df=gid_get001(hss)
print('')
print(df.tail())
df.to_csv('tmp\gid01.csv',index=False,encoding='gbk')

```

案例 7-6 运行后，在 tmp 目录下生成一个 gid01.csv 数据文件，可以用 Excel 打开，其截图如图 7-10 所示。

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	gid	gset	aplay	xtid	gplay	gid	jq	sq	rq	kend	tweek	tplay	tsell
2	581438	海晏	阿德里				-1	-1	0	1	星期五	2017/1/20	2017/1/20 16:45
3	632530	球会友谊	瓦勒伦	斯托曼			-1	-1	0	1	星期五	2017/1/20	2017/1/20 19:25
4	627923	非洲杯	特迪	民刚果			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
5	574559	法乙	阿弗尔	奥尔良			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
6	574563	法乙	阿弗尔	阿弗尔			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
7	574560	法乙	欧塞尔	布尔格			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
8	574565	法乙	尼姆	北顿			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
9	574567	法乙	尼姆	克雷斯			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
10	574568	法乙	斯特堡	图尔			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
11	574562	法乙	拉瓦勒	特鲁瓦			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
12	574566	法乙	尼斯	瓦朗谢			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
13	575926	德甲	特温特	赫拉克			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
14	579999	德甲	格拉夫	波博恩			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
15	579999	德甲	多蒙特	乌普斯			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
16	579999	德甲	法兰克福	法兰克福			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
17	579994	德甲	法兰克福	特林斯			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
18	579992	德甲	海尔蒙	奥格斯			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
19	579999	德甲	布吕姆	阿门堡			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
20	579997	德甲	瓦尔韦	埃门			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
21	627922	非洲杯	摩洛哥	多哥			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
22	596313	德甲	斯图加	科隆			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
23	575277	德甲	柏林	沙勒罗			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
24	577848	英超	阿森纳	切尔西			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
25	578408	德甲	维尔茨	汉堡			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
26	607022	德甲	拉斯帕	柏林赫			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
27	573989	德甲	巴斯蒂	尼斯			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
28	609755	英超	阿森纳	阿森纳			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
29	625950	中足联	阿森纳	阿森纳			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
30	631714	阿塞拜	阿森纳	阿森纳			-1	-1	0	1	星期五	2017/1/21	2017/1/20 23:55
31	629564	中足联	阿森纳	阿森纳			-1	-1	0	1	星期五	2017/1/21	2017/1/21 9:55
32	628778	墨西哥	阿森纳	阿森纳			-1	-1	0	1	星期五	2017/1/21	2017/1/21 10:55

图 7-10 gid 基本数据截图

由图 7-10 可以看出, `gid` 比赛数据大部分都成功提取了, 例如联赛名称、比赛 `id`、比赛时间、主队名称、客队名称等, 不过还是缺少几组数据, 包括:

- 主队代码 `id`、客队代码 `id`;
- 比赛得分情况, 即进球数、失球数、让球数。

这些数据我们会在稍后的案例中补充完整。

案例 7-6 代码不长, 结构也很简单, 只是在案例 7-5 的基础上补充了以下部分:

- `DataFrame` 变量 `df`、`Series` 变量 `ds` 的设置与初始化;
- `df` 数据的追加;
- 对于联赛、球队名称, 调用 `t.str_fltHtmHdr` 函数, 清理空白和乱码字符;
- 过滤并删除 `gid=-1` 的比赛数据;
- 采用 GBK 编码格式, 保存数据文件。

7.3.3 `gid` 比赛基本数据结构

在案例 7-6 的 `gid_get001` 函数定义前面的几行代码中, 有两行代码用于设置 `gid` 比赛的数据结构:

```
df=pd.DataFrame(columns=tfsys.gidSgn,dtype=str)
ds=pd.Series(tfsys.gidNil,index=tfsys.gidSgn,dtype=str)
```

以上代码在 `tfb_sys.py` 足彩系统参数模块中, 对 `gid` 数据结构采用了预定义的字段名称及字段初始化数值。使用初始化数值的一个好处是, 可以大体上规范各个字段的数据类型, 无需再对每一个数据列进行类型定义, 简化编程代码。

```
gidNil=['',' ',' ',' ',' ',' ',' ',' -1',' -1','0','0',' -1',' -1',' ',' ',' ']
gidSgn=['gid','gset','mplay','mtid','gplay','gtid','qj','qs','qr','kend','kwin','kwinrq','tweek','tplay','tsell']
```

其中, `gidSgn` 是 `gid` 比赛数据的字段名称, 其他参数说明如下。

- `gid`, 比赛场次的编码, 是系统唯一的 `id`。
- `gset`, `game-set` 的缩写, 联赛名称。
- `mplay`, `main-play` 的缩写, 主队名称。
- `mtid`, `main-team-id` 的缩写, 主队的编码 `id`。
- `gplay`, `guest-play` 的缩写, 客队名称。
- `gtid`, `guest-team-id` 的缩写, 客队的编码 `id`。

- qj, 进球的拼音缩写, 这个是国内行业惯例。
- qs, 失球的拼音缩写, 这个是国内行业惯例。
- qr, 让球的拼音缩写, 这个是国内行业惯例。本参数暂未使用, 供日后扩展使用。
- kend, key for end 的缩写, 1 代表比赛完结, 有些被取消的比赛其值一直是零。
- kwin, 比赛胜负情况, 3 为主队胜, 1 为平局, 0 为客队胜, 默认和取消的比赛数值是-1。
- kwinrq, 让球的比赛胜负情况, 3 为主队胜, 1 为平局, 0 为客队胜, 默认和取消的比赛数值是-1; 本参数暂未使用, 供日后扩展使用。
- tweek, time for week 的缩写, 比赛星期换算, 一般周末比赛最多。
- tplay, time for play 的缩写, 比赛日期。
- tsell, time for sell 的缩写, 彩票截止销售时间。

如图 7-11 所示是 500 彩票网站的比赛数据网页截图, 本书 gid 比赛基本数据文件的数据构成参考了 500 彩票网站的网页结构, 同时结合编程经验设计的。

胜平负/让球 胜平负 让球胜平负 五大联赛 1.60赔率以下 更多选择 已截止(0场) 定制 赛事回顾 2017-01-20									
编号	赛事	开赛时间	主队 VS 客队	让球	投注区			数据	平均赔率
胜				平	负				
星期五 2017-01-20 31场比赛已截止				0	1.49	3.85	5.05		
001	英超	16:50	[1] 悉尼FC 2:0 阿德莱 [9]	-1	2.54	3.50	2.22	析亚款	- -
031	球会友谊	19:30	瓦勒伦 3:2 斯托曼	0	1.44	4.20	5.10	析亚款	- -
				-1	2.30	3.70	2.36		
002	非洲杯	00:00	[2] 科特迪 2:2 刚果 [1]	0	1.52	3.16	6.45	析亚款	- -
				-1	3.10	2.85	2.20		
003	法乙	03:00	[15] 阿雅克 1:0 奥尔良 [20]	0	2.00	2.65	3.96	析亚款	- -
				-1	4.60	3.40	1.62		
004	法乙	03:00	[9] 阿弗尔 1:2 阿雅GF [11]	0	2.10	2.60	3.75	析亚款	- -
				-1	4.95	3.50	1.56		
005	法乙	03:00	[19] 欧塞尔 0:2 布尔格 [14]	0	2.07	2.70	3.62	析亚款	- -
				-1	4.95	3.55	1.55		
006	法乙	03:00	[13] 尼奥特 2:1 亚眠 [4]	0	2.59	2.60	2.79	析亚款	- -
				-1	6.45	4.00	1.38		
007	法乙	03:00	[6] 索肖 3:3 克莱蒙 [8]	0	1.88	2.65	4.55	析亚款	- -
				-1	4.20	3.20	1.73		
008	法乙	03:00	[5] 斯特堡 图尔 [17]	0	1.65	3.05	5.15	析亚款	- -
				-1	3.30	3.28	1.92		
009	法乙	03:00	[18] 拉瓦勒 1:0 特鲁瓦 [7]	0	2.52	2.62	2.85	析亚款	- -
				-1	6.30	3.90	1.40		
010	法乙	03:00	[3] 兰斯 0:0 瓦朗谢 [10]	0	1.57	3.18	5.61	析亚款	- -
				-1	3.34	2.96	2.03		

图 7-11 500 彩票网站的比赛数据网页截图

对比图 7-11 和 gid 的数据列组成, 可以发现, 网页的部分信息如编号、球队排名等数据, 在 gid 比赛基本数据文件中都没有收录。

此外,从图 7-11 中可以看到,500 彩票网站的比赛数据网页中,不仅提供了 gid 基本比赛数据,还提供了部分赔率数据。某些彩票网站在比赛网页中,甚至提供了最常用的十大博彩公司的数据,如必发、威廉、snai 等机构的赔率数据。在这些彩票网站只需抓取一个网页,就可获得 gid 比赛数据和 gdat 常用赔率数据,效率上快很多。

笔者采用的是多次提取模式,先抓取 gid 基本比赛数据,再根据 gid 比赛 id 编码,提取主流的赔率数据。

这样操作虽然效率低一点,但是在数据采集、未来扩展方面灵活很多,不仅可以提取更多的欧赔数据,而且可以提取亚赔、大小球及球队分析等各种数据。

7.3.4 案例 7-7: 提取比赛得分

案例 7-6 虽然提取了大部分的比赛数据,不过还是缺少几组数据,包括:

- 主队代码 id、客队代码 id;
- 比赛得分情况,包括进球数、失球数、让球数。

案例 7-7 的文件名是 zc707_gidget03xq.py,用于提取比赛得分情况,包括进球数、失球数和让球数,并生成 kwin 比赛胜负数据,核心代码如下:

```
#--2#
x20=bs.find_all('a',class_='score')
for xc,x in enumerate(x20):
    xss=x['href']
    kss=zstr.str_xmid(xss,'ju-','.sh')
    clst=x.text.split(':')
    #
    ds=df[df['gid']==kss]
    ds=df[df['gid']==kss]
    if len(ds)==1:
        inx=ds.index
        df['qj'][inx]=clst[0]
        df['qs'][inx]=clst[1]
        kwin=tft.fb_kwin4qnum(int(clst[0]),int(clst[1]))
        df['kwin'][inx]=str(kwin)
    #
```

```
print('\n@x',xc,'#inx',inx.values)
print('@x',xc,'#attrs',x.attrs)
print('@x',xc,'#x',x)
```

案例 7-7 中没有编写新函数，只是在 `gid_get001` 函数中追加了一组代码，为了以示区别，特意增加了一个#2 编码，表示本函数的第二组编码，这个也是基于工程一线的一种编程技巧，逐渐扩展来增加函数的功能。

第 2 组代码最后几个 `print` 语句用于输出节点 `x20` 保存的数据信息，正式版本需要屏蔽这些 `print` 语句，以免影响效率、干扰正式的输出信息。

案例 7-7 的部分输出信息是：

```
@x 28 #inx, [29]
@x 28 #attrs, {'target': '_blank', 'class': ['score'], 'href':
'http://odds.500.com/fenxi/shuju-629564.shtml'}
@x 28 #x, <a class="score" href="http://odds.500.com/fenxi/
shuju-629564.shtml" target="_blank">1:0</a>

@x 29 #inx, [30]
@x 29 #attrs, {'target': '_blank', 'class': ['score'], 'href':
'http://odds.500.com/fenxi/shuju-628778.shtml'}
@x 29 #x, <a class="score" href="http://odds.500.com/fenxi/
shuju-628778.shtml" target="_blank">1:0</a>

gid gset mplay mtid gplay gtid qj qs qr kend kwin kwinrq tweek
tplay          tsell
  26 609755 葡超 费雷拉 摩雷伦斯 0 2 0 1 0 -1 星期
五 2017-01-21 2017-01-20 23:55:00
  27 629563 中美杯 洪都拉 哥斯达 1 1 0 1 1 -1 星期
五 2017-01-21 2017-01-20 23:55:00
  28 631714 阿夏赛 图库曼 阿独立 0 0 0 1 1 -1 星期
五 2017-01-21 2017-01-20 23:55:00
  29 629564 中美杯 巴拿马 萨尔瓦 1 0 0 1 3 -1 星期
五 2017-01-21 2017-01-21 09:55:00
  30 628778 墨西联 韦拉克鲁 阿特拉 1 0 0 1 3 -1 星
期五 2017-01-21 2017-01-21 10:55:00
```

由输出结果可以看出，球队得分数据已经成功提取。

第 2 组节点数据 `x20`，提取代码只有如下一句：

选择的特征字符串是 `score`，这是对网页 HTML 源码进行多次测试得到的结果。如图 7-12 所示是对应的网页 HTML 源码截图，读者可以参考一下。

使用 `str xmid` 函数获取 `gid` 数据，并赋值给变量 `kss`：

将 `kss` 变量与 `df` 数据队列的 `gid` 数据列进行对比，找到正确的数据行 `ds`，再根据 `ds` 的 `index` (索引)，对正确的数据变量进行赋值；如果没有找到匹配的数据行 `ds`，或者找到的结果不只一个，则说明数据有问题，不会进行赋值操作：

比分数据是通过文本信息直接获取的，并转换为 `clst` 列表，再分别赋值给进球、失球对应的数据变量部分：

```
clst=x.text.split(':')

df['qj'][inx]=clst[0]
df['qs'][inx]=clst[1]
```

从文本信息直接提取数据的一个好处是，有时网页的编码结构发生变化，但显示文本并不一定改变，这样就无需修改网页抓取分析程序。

在后面的赔率数据提取部分，还会介绍更多通过文本信息提取数据的编程技巧。

得到进球数据后，程序调用 `tfb_tools` 模块的 `fb_kwin4qnum` 函数，生成 `kwin` 比赛胜负数据：

```
kwin=tfb.fb_kwin4qnum(int(clst[0]),int(clst[1]))
df['kwin'][inx]=str(kwin)
```

`tfb_tools` 模块的 `fb_kwin4qnum` 函数代码如下：

```
def fb_kwin4qnum(jq,sq,rq=0):
    if (jq<0)or(sq<0):return -1
    #
    jqk=jq+rq #or -rq
    if jqk>sq:kwin=3
    elif jqk<sq:kwin=0
    else:kwin=1
    #
    return kwin
```

`fb_kwin4qnum` 函数还可以计算让球情况下的比赛胜负数据，TFB 系统中的 `gid` 数据结构里面也预存了 `qr` 让球和 `kwinrq` 让球胜负数据的字段，不过目前暂时都没有使用。

为潜在的数据预留空间，也是软件工程一线的标准模式，有利于日后扩充系统，特别是目前内存、硬盘价格都非常低廉，用空间换取时间和提升开发效率是常用的工作模式，除非是一些对于储存空间等非常敏感的开发环境。

7.3.5 案例 7-8：提取球队 id 编码

案例 7-8 的文件名是 `zc708_gidget04team.py`，用来提取球队编码数据，核心代码如下：

```
#---3#
x20=bs.find_all('td',class_='left_team')
if (len(x20)==len(x10)):
    for xc,x in enumerate(x20):
        print('\n@4#x',xc,'#',x.a['href'])
        print(xc,'#',x.attrs)
        xss=x.a['href']
```

```

        xid=zstr.str_xmid(xss,'/team/', '/')
        df['mtid'][xc]=xid

#---4#
x20=bs.find_all('td',class_='right_team')
if (len(x20)==len(x10)):
    for xc,x in enumerate(x20):
        print('\n@4#x',xc,'#',x.a['href'])
        print(xc,'#',x.attrs)
        xss=x.a['href']
        xid=zstr.str_xmid(xss,'/team/', '/')
        df['gtid'][xc]=xid

```

案例 7-8 与案例 7-7 类似,在案例 7-8 的 `gid_get001` 函数中追加了两组代码,分别是代码组#3 和代码组#4。

因为网页的 HTML 源码用字符串 `left_team` 表示主队、用字符串 `right_team` 表示客队,因此两组程序除了特征码不同,其他结构完全相同。

案例 7-8 的部分运行结果如下:

```

@4#x 30 # http://liansai.500.com/team/1891/
30 # {'class': ['right_team']}

        gid gset mplay mtid gplay gtid qj qs qr kend kwin kwinrq tweek
tplay          tsell
26 609755 葡超 费雷拉 655 摩雷伦斯 1134 0 2 0 1 0 -1
星期五 2017-01-21 2017-01-20 23:55:00
27 629563 中美杯 洪都拉 796 哥斯达 1110 1 1 0 1 1 -1
星期五 2017-01-21 2017-01-20 23:55:00
28 631714 阿夏赛 图库曼 42 阿独立 8 0 0 0 1 1 -1 星
期五 2017-01-21 2017-01-20 23:55:00
29 629564 中美杯 巴拿马 5361 萨尔瓦 1322 1 0 0 1 3 -1
星期五 2017-01-21 2017-01-21 09:55:00
30 628778 墨西联 韦拉克鲁 121 阿特拉 95 1 0 0 1 3 -1
星期五 2017-01-21 2017-01-21 10:55:00

```

从案例 7-8 的输出信息可以发现, `x.attrs` 输出的信息基本都是无用信息,真正有用的数据即球队的编码 `id` 隐藏在链接地址中,对应的网页信息如图 7-13 所示。

```

<td class="left_team">
  <span class="gray">[9]</span>
  <a href="http://liansai.500.com/team/1885/" target="_blank" title="韦拉克鲁斯">韦拉克鲁斯</a>
</td>
<td>
  <a class="score" href="http://odds.500.com/fenxi/shuju-628778.shtml" target="_blank">1:0</a>
<td class="right_team">
  <a href="http://liansai.500.com/team/1891/" target="_blank" title="阿特拉斯">阿特拉</a>
<span class="gray">[12]</span>
</td>

```

图 7-13 球队 id 信息网页截图

对于球队编码 id 及其他的部分隐含信息，都可以使用这种方法轻松抓取。500 彩票网站作为上市公司，网站数据还算是比较开放的，除了后面的赔率数据。其采用的是瀑布流动态模式，基本上分析所需的数据都能够直接获取。

其他网站往往会使用 JavaScript 等内置函数或者嵌入网页模式隐含真正的数据，这个情况在不少视频、电影下载网站经常会碰见。

在案例 7-8 中，选择字符串 left_team、right_team 作为特征码，使用 zt.str_xmid 函数提取球队 id，这些都是常规的编程技巧，读者多编写一些网页抓取分析程序，多看看网页 HTML 源码，慢慢积累，就可以熟能生巧。

案例 7-8 中需要注意的是，在设置球队代码数据时，没有使用 index（索引）：

```
df['gtid'][xc]=xid
```

因为在案例 7-8 中，没有可以提取 gid 数据的部分，所以强行使用序号 xc 作为 df 变量的下标。

此外，为了防止网页乱码，出现下标错误，在 for 循环前增加了一个长度检查语句，使 x20 节点与 x10 节点的数目匹配：

```

if (len(x20)==len(x10)):
    for xc,x in enumerate(x20):

```

至此，从比赛数据网页提取 gid 比赛数据的编程工作就全部完成了，将以上代码合并整理，作为一个独立的函数 fb_gid_get4htm 保存在 tfb_tools.py 模块中，读者直接调用即可。

本章的几个案例，从多个角度、使用多种不同的技巧，分析 500 彩票网站比赛网页的信息，提取需要的 gid 比赛数据。

这种方式可能不是最简捷的，甚至有些烦琐，特别是针对 500 彩票网站的案例而言。

不过网络情况千变万化，各个网站的网页数据各不相同，读者在实际工作中所需要面对的情况也非常复杂，笔者采用这种分步模式，从多个角度来分析不同的网页，希望能够给读者传授更多的实战技巧。

7.3.6 案例 7-9：抓取历年比赛数据

前面已经学习了从 500 彩票网站抓取比赛数据网页，并从网页 HTML 源码中提取 gid 比赛数据。

案例 7-9 的文件名是 `zc709_gidall.py`，汇总前面学过的知识，编写一个真正的商业级别的提取球队比赛数据的程序，核心代码如下：

```
xtfb=tft.fb_init()
tfsys.gidsFN='tmp/gid01.csv';
zsys.web_get001txtFg=True
#
tim0str='2010-01-01'
tim0=arrow.get(tim0str)
tn=arrow.now()-tim0
print('tn,',tn)
#
timStr,nday='',2
tfsys.xnday_down=nday
fb_gid_get_nday(xtfb,timStr,fgExt=False)
#
```

输出信息如下：

```
tn, 2594 days, 3:24:24.084816
0 # /tfbDat/xhtm/ghtm/2017-02-07.htm
    /tfbDat/xhtm/ghtm/2017-02-07.htm 318707
1 # /tfbDat/xhtm/ghtm/2017-02-06.htm
    /tfbDat/xhtm/ghtm/2017-02-06.htm 214844

      gid gset mplay mtid gplay gtid qj qs qr kend kwin kwinrq tweek
tplay      tsell
  14 596642 德乙 斯图加 1195 杜塞尔 781 2 0 0 1 3 -1
1 2017-02-07 2017-02-06 23:55:00
  15 607042 西甲 格拉纳 4607 拉斯帕 965 1 0 0 1 3 -1
1 2017-02-07 2017-02-06 23:55:00
  16 574598 法乙 瓦朗谢 2331 阿弗尔 982 0 0 0 1 1 -1
1 2017-02-07 2017-02-06 23:55:00
  17 609772 葡超 菲伦斯 2923 里奥阿 1002 2 1 0 1 3 -1
```

```
1 2017-02-07 2017-02-06 23:55:00
   18 609771 葡超 布拉加 719 埃斯托 1616 1 1 0 1 1 -1
1 2017-02-07 2017-02-06 23:55:00
```

案例 7-9 的主程序很简单，流程如下。

- 调用 `fb_init` 函数，初始化系统参数 `xtfb`，这也是 TFB 极宽足彩量化分析软件的标准流程。
- 设置 `gid` 文件名，将 `zsys.web_get001txtFg` 参数设为 `True`，表示重新下载网页文件，即使硬盘中已经存在该网页文件。
- 计算系统首场比赛日期（2010-01-01）距目前日期的时间。笔者测试时是 2594 天，读者运行的具体时间不同，因此会有所差异，属于正常情况。
- 设置 `nday` 为 2，抓取最近两天的 `gid` 比赛数据，生成 `timstr` 变量。如果要抓取全部历史数据，设置 `nday` 为 `tn` 即可。
- 把 `nday` 的数值赋值给全局变量 `tfsys.xnday_down`。
- 调用 `fb_gid_get_nday` 函数抓取网页，提取 `gid` 数据，并保存为 `gid01.csv` 文件。
- 调用 `fb_gid_get_nday` 函数时，`fgExt` 参数设置为 `False`，实盘时应该设为 `True`。

本案例因为只是演示现在的 `gid` 数据，所以 `FgExt` 参数设置为 `False`。

案例 7-9 的核心是 `fb_gid_get_nday` 函数，原本属于 `tfb_tools.py` 模块，为了便于说明，笔者特意在案例 7-9 主程序中复制了一份，函数代码如下：

```
def fb_gid_get_nday(xtfb,timStr,fgExt=False):
    if timStr=='':ktim=xtfb.tim_now
    else:ktim=arrow.get(timStr)
    #
    nday=tfsys.xnday_down
    for tc in range(0,nday):
        xtim=ktim.shift(days=-tc)
        xtimStr=xtim.format('YYYY-MM-DD')
        #print('\nxtim',xtim,xtim<xtfb.tim0_gid)
        #
        xss=str(tc)+'#','+xtimStr+',@'+ zt.get_fun_nam()
        zt.f_addLog(xss)
        if xtim<xtfb.tim0_gid:
            print('#brk;')
            break
    #
```

```

fss=tfsys.rghtm+xtimStr+'.htm'
uss=tfsys.us0_gid+xtimStr
print(timStr,tc,'#',fss)
#
htm=zweb.web_get001txtFg(uss,fss)
if len(htm)>5000:
    df=tft.fb_gid_get4htm(htm)
    if len(df['gid'])>0:
        tfsys.gids=tfsys.gids.append(df)
        tfsys.gids.drop_duplicates(subset='gid', keep='last',
inplace=True)
#
if fgExt:tft.fb_gid_getExt(df)
#if fgExt:tft.fb_gid_getExtPool(df)
#
if tfsys.gidsFN!='':
    print('')
    print(tfsys.gids.tail())
    tfsys.gids.to_csv(tfsys.gidsFN,index=False,encoding='gb18030')

```

如图 7-14 所示是 fb_gid_get_nday 函数的流程图。

fb_gid_get_nday 函数使用了 arrow 新一代时间模块，相对比较简单，根据日期和制定的天数 nday 循环抓取网页数据，并追加到全局变量 tfsys.gids 中，有关的技巧如下。

- xtim=ktim.shift(days=-tc)，使用 arrow 说明模块的 shift 内置时间位移函数，大大简化了相关的时间运算代码。
- tfsys.gids.drop_duplicates(subset='gid', keep='last', inplace=True)，数据追加后，自动去重，保留最新数据。
- tfsys.gids.to_csv(xtfb.gidsFN,index=False,encoding='gb18030')，按 GB18030 编码格式保存数据文件。

保存程序中的数据文件时，采用的是 GB18030 编码格式，如果按 GBK 编码格式有时会出错，UTF 原本是最好的选择，也是 Python 3 的默认编码格式，不过作为数据源的 500 彩票网站采用的是 GBK 模式，所以本案例采用了 GBK 的超集 GB18030 编码格式。

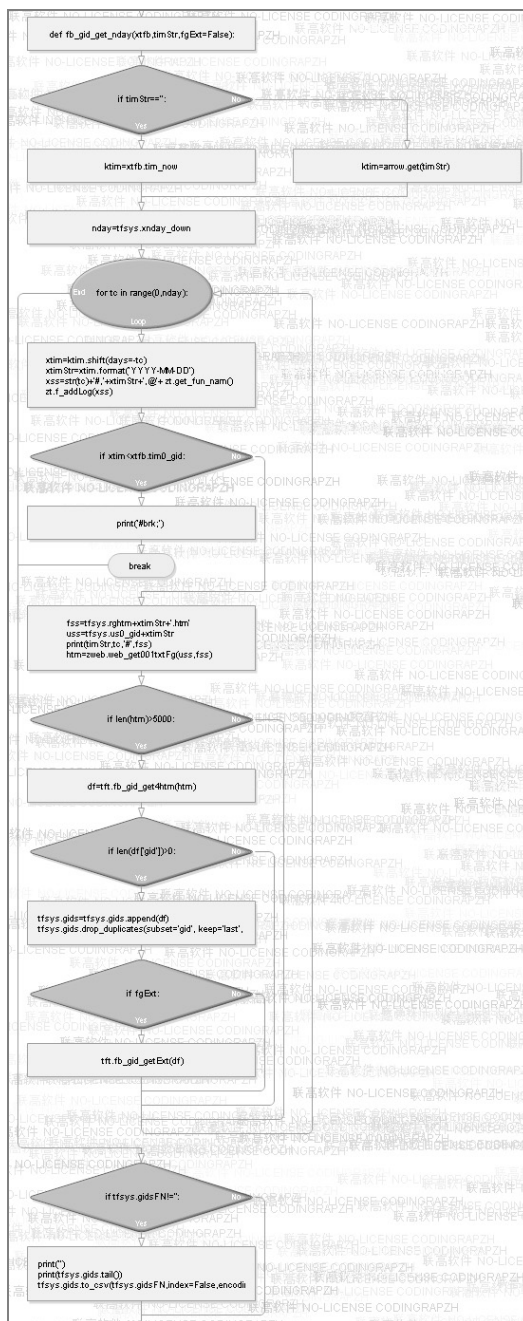


图 7-14 fb_gid_get_nday 函数流程图

7.3.7 案例 7-10：流程图工具与 Python

案例 7-9 中的流程图是使用共享工具软件 CodinGraph 绘制的，这个软件绘制流程图很方便，只要复制相关的函数代码，软件就会自动绘制对应的流程图。

对于 Python 程序而言，因为代码块采用的是空格缩进模式，与其他编程语言的“{}”符号和“begin、end”模式完全不同，因此在使用前，需要对有关代码块进行处理，增加相关的“{}”符号，表示代码区块的起始与结束。

案例 7-10 的文件名是 `zc710_codgr.py`，代码是修改过的 `fb_gid_get_nday` 函数，只是为了说明如何使用 CodinGraph 软件正确地绘制 Python 语言的流程图。

案例 7-10 的代码是不能运行的，请读者注意，代码当中的“{}”符号是笔者后加的，请注意与案例 7-9 中的 `fb_gid_get_nday` 函数进行对比。

案例 7-10 的全部代码如下：

```
def fb_gid_get_nday(xtfb,timStr,fgExt=False):
    if timStr=='':
        ktim=xtfb.tim_now
    else:
        ktim=arrow.get(timStr)
    #
    nday=tfsys.xnday_down
    for tc in range(0,nday):
    {
        xtim=ktim.shift(days=-tc)
        xtimStr=xtim.format('YYYY-MM-DD')
        #print('\nxtim',xtim,xtim<xtfb.tim0_gid)
        #
        xss=str(tc)+'#,'+xtimStr+',@'+ zt.get_fun_nam()
        zt.f_addLog(xss)
        if xtim<xtfb.tim0_gid:
        {
            print('#brk;')
            break
        }

        fss=tfsys.rghtm+xtimStr+'.htm'
```

```

uss=tfsys.us0_gid+xtimStr
print(timStr,tc,'#',fss)
#
htm=zweb.web_get001txtFg(uss,fss)
if len(htm)>5000:
{
    df=tft.fb_gid_get4htm(htm)
    if len(df['gid'])>0:
    {
        tfsys.gids=tfsys.gids.append(df)
        tfsys.gids.drop_duplicates(subset='gid', keep='last',
inplace=True)

        #
        if fgExt:
            tft.fb_gid_getExt(df)
            #if fgExt:tft.fb_gid_getExtPool(df)
    }
}
#
if tfsys.gidsFN!='':
{
    print('')
    print(tfsys.gids.tail())
    tfsys.gids.to_csv(xtfb.gidsFN,index=False,encoding='gb18030')
}

```

7.3.8 实盘技巧

案例 7-9 中的 `fb_gid_get_nday` 函数与笔者实际抓取 `gid` 历史数据的函数基本相同，可以说是商业级别的代码程序。

唯一不同的是下面这行代码：

```

if fgExt:tft.fb_gid_getExt(df)
#if fgExt:tft.fb_gid_getExtPool(df)

```

案例 7-9 抓取赔率网页使用的是 `fb_gid_getExt` 函数，是标准的 Python 函数，实

盘中为了提高效率,使用的是 `fb_gid_getExtPool` 函数,通过进程池并发模式,大约可以提高 1~3 倍的速度,具体细节参见稍后的案例分析。

进行实盘抓取时,笔者已经预先下载了有关的 `gid` 历史比赛数据,一般是追加模式,读者只要运行更新的部分就行了。`zsys.web_get001txtFg` 函数的参数为 `True`,表示重新下载网页文件,采用最新的赔率数据,覆盖原来的网页文件,一般 `nday` 设置为 2 天。

如果希望提取不同时段赔率变化数据,可以在文件名中增加小时、分钟参数,不覆盖原来的网页文件,另外一种方法就是自己分析 500 彩票网站的赔率变化代码,直接提取变化的赔率数据,类似金融股票的 `tick` 数据,这样更加全面,而且没有遗漏。

如果需从零基础开始自己抓取全部数据,由于网络情况复杂,有时会出现断线等下载出错情况,则需要进行多轮下载。

- 前面 1~2 轮下载,将 `zsys.web_get001txtFg` 函数的参数设置为 `True`,表示重新下载网页。
- 后面几轮下载属于补漏阶段,将 `zsys.web_get001txtFg` 函数的参数设置为 `False`,以提高效率。

抓取全部数据还需要对本程序进行性能方面的优化,不然近七万场比赛的数据、两千六百天的网页数据,在网络畅通的前提下,理论上单机需要抓取 5~6 个小时。

采用目前 Python 最好的并发模块库 `concurrent.futures`,通过 `ProcessPool` 进程池技术,大约可以提速 3~5 倍,1~2 个小时即可完成下载。

笔者采用的是极宽公司自行开发的 `MTrd3.0` 专利并发算法,可以提速 30~50 倍,只用一台 `i7` 笔记本电脑,不到一个小时就完成了历年 `gid` 比赛数据的抓取。

真正能够体现 `MTrd3.0` 技术性能优越的地方是对历史赔率数据的抓取,7 万场比赛就是 7 万个网页,原来都要抓取数日,现在几个小时就可以全部抓取完成。

这次足彩历史数据的抓取,一方面验证了 `zwPython` 开发平台作为工业级开发平台的稳定性,另一方面也说明了 500 彩票网站作为数据源的稳定性。

做网页数据采集的程序员都知道,很多数据接口,包括百度的 `API`,甚至一些收费的 `API`,对于数据采集的频率都有限制,同一个 `ID` 或者 `IP`,一个小时最多只有几千次的数据采样频率,足彩行业有些甚至每天限制在 200~300 次,这个频率对于采集历史数据是远远不够的。

7.3.9 案例 7-11：进程池并发运行

Python 语言因为内部的 GIL 问题，对于并发、多进程和多线程的支持一直很弱，包括最新的 Python 3.6 携程模式。

Python 的 map 函数是笔者见过的最优雅的并发函数原型，比 MATLAB 的 gfor 语句还要简单；理论上，只需修改 map 底层函数，即可完成 Python 的并发开发，包括 CUDA 等 GPU 加速。有关并发、多进程的开发，属于相对比较专业的领域，在此不做过多探讨。

初学者对于本节内容暂时无法理解，属于正常现象，可以保留问题，待全书学习完毕或者水平提高再深入学习。

案例 7-11 通过简单的对比测试，介绍 Python 的并发编程，对于性能方面的提升，主要代码如下：

```
fss='dat/gid50.dat'
fss='dat/gid20.dat'
tst_pool(fss)
```

案例 7-11 通过自定义的 tst_pool 函数进行对比测试，主流程只有两行代码，其中的 gid20、gid50 都是标准的 gid 比赛数据文件，主要用于提供 gid 参数，gid20、gid50 两个数据文件分别有 20 场、50 场比赛的 gid 数据。

案例 7-11 的重点在 tst_pool 测试函数，函数代码较长，已经进行了分组编码，下面逐一进行讲解。

tst_pool 测试函数的第 1 组代码如下：

```
#1---set
tfsys.rghtm,tfsys.rxhtm,tfsys.rhtmOuzhi='tmp/','tmp/','tmp/'
tfsys.rxdatt,tfsys.rgdatt='tmp/','tmp/'
tfsys.xnday_down=2
zsys.web_get001txtFg=True
```

第 1 组代码主要是参数设置，重点是把所有的输出目录都改为 tmp 临时目录，避免对 tfbDat 数据造成干扰。

tst_pool 测试函数的第 2 组代码如下：

```
#2
df=pd.read_csv(fgid,index_col=False,dtpe=str,encoding='gbk')
print(df.tail())
```

第2组代码根据调用时的文件名参数 fgid，读取 gid 比赛数据文件。

tst_pool 测试函数的第3组、第4组代码如下：

```
#3
tim0=arrow.now()
tft.fb_gid_getExt(df)
tn1=zt.timNSec(' ',tim0,True)
#
#4
tim0=arrow.now()
tft.fb_gid_getExtPool(df)
tn2=zt.timNSec(' ',tim0,True)
#
```

第3组和第4组代码内容大体相同，分别调用 fb_gid_getExt 标准赔率数据下载函数和进程池的 fb_gid_getExtPool 并行赔率数据下载函数，记录相关的运行时间。

tst_pool 测试函数的第5组代码如下：

```
#5---
print('\n#5')
print('fb_gid_getExt,tim,',tn1)
print('fb_gid_getExtPool,tim,',tn2)
```

第5组代码用于输出相关的运行时间数据。

在函数的尾部，有笔者 i7 笔记本电脑的运行结果，单位是秒，如图 7-15 所示。

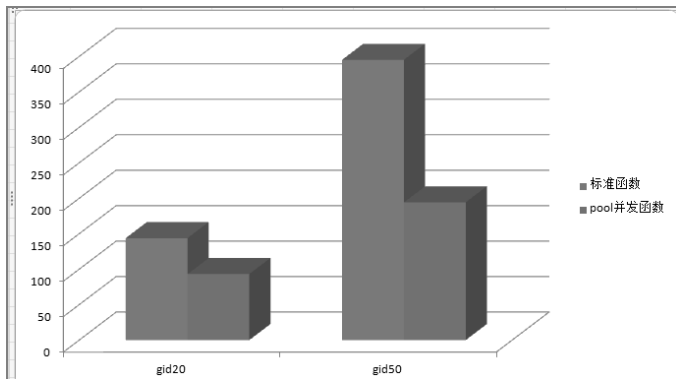


图 7-15 并行运行速度对比

其中，gid20 数据文件的运行结果是：

```
@gid20
```

```
fb_gid_getExt,tim, 143.7
fb_gid_getExtPool,tim, 93.07
```

gid50 数据文件的运行结果是：

```
@gid50
fb_gid_getExt,tim, 395.24
fb_gid_getExtPool,tim, 194.29
```

换算一下，进程池并发编程，性能提升分别如下：

- gid20 数据文件， $143.7/93.07 \times 100 = 154.4\%$ ；
- gid50 数据文件， $395.24/194.29 \times 100 = 203.4\%$ 。

性能方面分别有 1.5~2 倍的速度提升，这是因为测试函数不仅抓取网页，还对网页数据进行解析，提取相关的赔率数据，属于 IO 和 CPU 并重的程序。

笔者实测过，纯网页抓取可以提速 3 倍，再提速就很困难，资料检索表明，这是因为 Python 并发技术的并发引擎需要预热和调度，预热和调度时间与运行时间的比例是 1:3，具体细节太过于学术化，读者自行参考。

下面看看两个调用函数的代码，fb_gid_getExt 标准赔率数据下载函数的代码如下：

```
def fb_gid_getExt(df):
    dn9=len(df['gid'])
    for i, row in df.iterrows():
        #xtfb.kgid=row['gid']
        #xtfb.bars=row
        fb_gid_getExt010(row.values)
        #
    print(zsys.sgnSP8,i,'/',dn9,'@ext')
```

采用进程池的 fb_gid_getExtPool 并行赔率数据下载函数的代码如下：

```
def fb_gid_getExtPool(df,nsub=5):
    pool=ThreadPoolExecutor(max_workers = nsub)
    xsubs = [pool.submit(fb_gid_getExt010,x10) for x10 in df.values]
    #
    for xsub in as_completed(xsubs):
        fss=xsub.result(timeout=20);
        print('@fb_gid_getExtPool,',fss)
```

两个函数都已经高度简化，便于初学者参考学习。

7.4 批量抓取足彩网页数据实盘教程

足彩大数据分析仅有 gid 基本比赛数据是远远不够的，其中最重要的赔率数据还没有，本节通过相关案例，讲解抓取足彩赔率数据网页，并从中提取相关的赔率数据。

7.4.1 案例 7-12：批量抓取赔率数据

案例 7-12 的文件名是 zc712_xdat001.py，介绍批量抓取足彩赔率数据。

足彩赔率数据的抓取和分析采用逆向工程模式，先整体再全局，和前面的网络 gid 比赛数据的抓取和分析恰好相反。

这样做的一个原因就是，案例 7-12 与案例 7-9 几乎完全一致，只是调用的参数不同，其核心代码如下：

```
tfsys.rghtm,tfsys.rxhtm,tfsys.rxdat,tfsys.rhtmOuzhi='tmp/','tmp/','tm
p/','tmp/'
timStr,nday='',2
tfsys.xnday_down=nday
fb_gid_get_nday(xtfb,timStr,fgExt=True)
```

案例 7-12 与案例 7-9 一样，也是调用 fb_gid_get_nday 批量网页足彩数据抓取函数，只是有两个地方不同。

第一个不同之处是：所有的数据目录都改为 tmp 临时目录，避免干扰正式的 tfbDat 足彩数据包。

```
tfsys.rghtm,tfsys.rxhtm,tfsys.rxdat,tfsys.rhtmOuzhi='tmp/','tmp/','tm
p/','tmp/'
```

正常情况下，这些目录的位置如下。

- rghtm, gid 网页保存目录，默认路径是\tfbDat\xhtm\ghtm\。
- rxhtm, 赔率网页保存目录，默认路径是\tfbDat\xhtm\。
- rhtmOuzhi, 欧赔网页保存目录，默认路径是\tfbDat\xhtm\htm_oz\。
- xdat, 提取后的欧赔数据保存目录，默认路径是\tfbDat\xdat\。

第二个不同之处是，在调用 fb_gid_get_nday 函数时，参数 fgExt 的值为 True，表示下载赔率数据：

```
fb_gid_get_nday(xtfb,timStr,fgExt=True)
```

在 `fb_gid_get_nday` 函数中，相关的代码是：

```
if fgExt:tft.fb_gid_getExt(xtfb,df)
```

如果参数 `fgExt` 的值为 `True`，就调用 `tfb_tools` 模块的 `fb_gid_getExt` 扩展下载函数，下载赔率等扩展信息网页。

为了便于学习和理解，本节使用的都是 `fb_gid_getExt` 标准赔率网页抓取函数，没有涉及进程池并行技术，请读者注意。

7.4.2 fb_gid_getExt 扩展网页下载函数

`fb_gid_getExt` 扩展网页下载函数位于 `tfb_tools` 模块，函数代码是：

```
def fb_gid_getExt(df):
    dn9=len(df['gid'])
    for i, row in df.iterrows():
        #xtfb.kgid=row['gid']
        #xtfb.bars=row
        fb_gid_getExt010(row.values)
        #
        print(zsys.sgnSP8,i,'/',dn9,'@ext')
```

`df` 是函数的输入参数，一般是当天的比赛数据，重点是 `gid` 比赛的 `id` 代码，以便合成赔率网页地址。

如果读者另外编写程序，读入 `gid2017` 总的 `gid` 比赛数据，也可以采用这种模式，批量下载扩充网页数据。

`fb_gid_getExt` 函数使用了如下编程技巧：

```
for i, row in df.iterrows():
```

它通过迭代的方式，访问 `Pandas` 变量 `df` 中的全部数据，无需知道 `df` 的大小。

7.4.3 bars 节点数据包与 pools 彩票池

在 `fb_gid_getExt` 函数中，有几句被屏蔽的代码：

```
#xtfb.kgid=row['gid']
#xtfb.bars=row
```


以上代码中使用了 `bars` 节点数据包，虽然函数为了配合实盘的进程池并行技术对代码进行了修改，屏蔽了 `bars` 节点数据包的使用，但不影响我们对 `bars` 节点数据包这个概念的学习。

`fb_gid_getExt` 函数使用的另外一个编程技巧是：

```
xtfb.bars=row
```

这里的变量 `bars` 表示一个节点的所有数据，源自 `tfb_sys` 模块的 `zTopFootball` 类定义：

```
self.bars=None
```

变量 `bars` 又称节点数据包，类似量化系统中常用的数据变量，在 TFB 足彩量化分析系统中，我们借用了 `bars` 这个概念并进行了扩展。

此外，我们还借用了量化中的 `pools` 股票池的概念来保存多个足彩数据，在 TFB 系统中称为彩票池，参见 `tfsys` 的相关代码：

```
self.poolInx=[]
self.poolDay=pd.DataFrame(columns=poolSgn)
#pool.all
self.poolTrd=pd.DataFrame(columns=poolSgn)
self.poolRet=pd.DataFrame(columns=retSgn)
```

在 TFB 系统中，彩票池包括：

- `poolInx`，比赛索引数据，只有 `gid` 编码和 1 天的数据；
- `poolDay`，各场比赛的赔率数据，只有 1 天的数据；
- `poolTrd`，总的交易数据，包括多日 `gid` 数据的增强版本，增加了指定的收盘赔率数据；
- `poolRet`，每天的回报率记录，包括多日的。

扩展后的 `bars` 节点数据包没有固定的结构，原始定义为 `None`。虽然定义是 `None`（空值），但因为没有类型限制，所以成了万能变量，任何数据都能保存，无论是单一的数值，还是批量的数据矩阵、列表。

在案例 7-12 中，`bars` 节点数据包保存的是单个比赛的 `gid` 数据，包括 `gid` 比赛编码、比赛时间、球队名称、进球、失球等数据。

在后面的案例中，我们还会使用 `bars` 节点数据包，针对同一个 `gid` 的单场比赛，保存多个庄家的赔率数据。

`bars` 节点数据包的概念有些抽象，初学者可能不好理解，可以先做个记号，等

全书看完，再回头深入研究。

7.4.4 抓取扩展网页

在本案例中，`fb_gid_getExt` 函数本身只是设置参数，再调用 `fb_gid_getExt010` 函数抓取网页。

`fb_gid_getExt010` 函数的代码如下：

```
def fb_gid_getExt010(x10):
    bars=pd.Series(x10,index=tfsys.gidSgn,dtype=str)
    gid=bars['gid']
    #
    fss=tfsys.rhtmOuzhi+gid+'_oz.htm'
    uss=tfsys.us0_extOuzhi+gid+'.shtml';#print(uss)
    htm=zweb.web_get001txtFg(uss,fss) #zt.zt_web_get001txtFg
or (fsiz<5000):
    #
    fxdat=tfsys.rxdat+gid+'_oz.dat'
    fsiz=zt.f_size(fxdat);#print(zsys.sgnSP4,'@',fsiz,fxdat)
    #
    #print('xtfb.bars',xtfb.bars)
    if (fsiz<1000)or(tfsys.xnday_down<10):
        fb_gid_getExt_oz4htm(htm,bars,ftg=fxdat)

    '''
    #
    fss=xtfb.rhtmYazhi+xtfb.kgid+'_az.htm'
    uss=xtfb.us0_extYazhi+xtfb.kgid+'.shtml'
    #
    fss=xtfb.rhtmShuju+xtfb.kgid+'_sj.htm'
    uss=xtfb.us0_extShuju+xtfb.kgid+'.shtml'
    '''

    return fxdat
```

`fb_gid_getExt010` 函数的代码很简单，流程如下。

- 根据 gid 合成 uss 下载网址、fss 网页文件名和 fxdat 赔率数据文件名。
- 调用 web_get001txtFg 函数，按 uss、fss 下载网页并保存。
- 如果网页长度大于 1000 字节，则调用 fb_gid_getExt_oz4htm 函数提取赔率数据。

需要注意的是，函数最后的几行注释表示 500 彩票网站的亚洲赔率网页地址和数据分析地址，参见 tft_sys 模块的 zTopFootball 类定义代码：

```
self.us0_extOuzhi='http://odds.500.com/fenxi/ouzhi-'
self.us0_extYazhi='http://odds.500.com/fenxi/yazhi-'
self.us0_extShuju='http://odds.500.com/fenxi/shuju-'
```

在 500 彩票网站赔率数据网页的顶部，都有一个共同的导航条，如图 7-16 所示。



图 7-16 比赛数据导航条

单击图 7-16 所示的比赛数据导航条，可以获得更多的比赛数据，如投注分析、让球指数、大小指数、比分指数、走势分析和技术统计等。

虽然欧赔、亚赔和数据分析的网页初始地址及网页保存目录，我们都进行了预设，但目前 TFB 暂时只使用了欧赔数据，其他暂时保留，作为日后扩展使用。

7.5 足彩赔率数据抓取

7.5.1 gid 与赔率数据网页

前面讲过 gid 比赛的 id 代码与其他数据网页的关系，以 gid 为 629388 的比赛为例，对应的欧赔、亚赔、数据分析网页分别是：

- 欧洲赔率，<http://odds.500.com/fenxi/ouzhi-629388.shtml>；
- 亚洲赔率，<http://odds.500.com/fenxi/yazhi-629388.shtml>；
- 分析网页，<http://odds.500.com/fenxi/shuju-629388.shtml>。

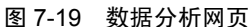
打开以上网页，分别如图 7-17～图 7-19 所示。

<div> <div>德黑兰独立</div> <div> <div>500</div> <div>17亚冠杯资格赛3</div> <div>比赛时间2017-02-07 23:30</div> </div> <div>多哈萨德</div> </div>												
VS												
<div> 数据分析 投注分析 百家欧赔 让球指数 亚盘对比 大小指数 比分指数 走势分析 技术统计 </div>												
序选	赔率公司	定制	即时欧赔			即时赔率			返还率	即时盈利		
			胜	平	负	胜	平	负	值	胜	平	负
1	竞彩官方		2.05	3.30	2.95	43.18%	26.82%	30.00%	88.51%	0.77	0.96	0.98
			2.05	3.30	2.95	43.18%	26.82%	30.00%	88.51%	0.77	0.96	0.98
2	威廉希尔		2.50	3.00	2.62	35.67%	29.90%	34.23%	89.69%	0.94	0.88	0.87
			2.45	3.10	2.62	36.69%	29.00%	34.31%	89.89%	0.92	0.90	0.87
3	澳门		2.45	3.05	2.62	36.52%	29.33%	34.15%	89.47%	0.92	0.89	0.87
			2.35	3.25	2.60	38.07%	27.53%	34.41%	89.46%	0.88	0.95	0.87
4	立博		2.30	3.30	3.00	40.59%	28.29%	31.12%	93.36%	0.86	0.96	1.00
			2.50	3.10	2.90	37.47%	30.22%	32.31%	93.68%	0.94	0.90	0.97
5	Bet365		2.50	3.20	2.50	35.96%	28.09%	35.96%	89.89%	0.94	0.93	0.83
			2.60	3.10	2.80	36.14%	30.31%	33.56%	93.06%	0.97	0.90	0.93
6	Interwetten		2.50	3.05	2.65	36.19%	29.67%	34.14%	90.48%	0.94	0.89	0.88
			2.45	3.05	2.70	36.89%	29.63%	33.48%	90.38%	0.92	0.89	0.90
7	SNAI		2.40	3.15	2.70	37.72%	28.74%	33.53%	90.54%	0.90	0.92	0.90
			2.45	3.10	2.70	37.07%	29.30%	33.64%	90.82%	0.92	0.90	0.90
8	皇冠		2.23	3.25	2.74	40.00%	27.45%	32.55%	89.20%	0.84	0.95	0.91
			2.40	3.25	2.65	37.82%	27.93%	34.25%	90.77%	0.90	0.95	0.88
9	易胜博		2.42	3.00	2.80	37.44%	30.20%	32.36%	90.60%	0.91	0.88	0.93
			2.42	3.00	2.80	37.44%	30.20%	32.36%	90.60%	0.91	0.88	0.93
10	伟德		2.30	3.25	3.00	40.41%	28.60%	30.98%	92.95%	0.86	0.95	1.00
			2.55	3.20	2.75	36.71%	29.25%	34.04%	93.61%	0.96	0.93	0.92

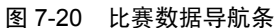
图 7-17 欧赔网页

<div> <div>德黑兰独立</div> <div> <div>500</div> <div>17亚冠杯资格赛3</div> <div>比赛时间2017-02-07 23:30</div> </div> <div>多哈萨德</div> </div>												
VS												
<div> 数据分析 投注分析 百家欧赔 让球指数 亚盘对比 大小指数 比分指数 走势分析 技术统计 </div>												
序选	赔率公司	定制	即时盘口				变化时间	初始盘口				变化时间
			水	盘	水	更多		水	盘	水		
1	澳门		0.800	平手	0.960		02-07 13:51	0.800	平手	0.960	02-07 13:51	主客同
2	Bet365		1.150	平手/半球	0.720		02-07 14:00	1.000	平手/半球	0.850	02-06 20:10	主客同
3	皇冠		1.130	平手/半球	0.700		02-07 14:00	0.990	平手/半球	0.830	02-06 20:01	主客同
4	易胜博		0.860	平手	1.060		02-07 07:29	0.860	平手	1.060	02-07 07:29	主客同
5	伟德		0.860	平手 降	1.030	3 ▼	02-07 02:17	1.010	平手/半球	0.860	02-06 20:45	主客同
6	Pinnacle平博		0.793	平手 降	1.050	5 ▼	02-07 13:24	1.000	平手/半球	0.840	02-06 20:23	主客同
7	利记		0.840	平手 降	1.000	1 ▼	02-07 06:57	0.990	平手/半球	0.850	02-06 20:00	主客同
8	Unibet (优胜客)		1.170	平手/半球	0.640	1 ▼	02-07 04:02	0.930	平手/半球	0.810	02-06 20:48	主客同
9	Mansion88 (明升)		1.163	平手/半球	0.700		02-07 12:14	1.000	平手/半球	0.840	02-06 20:24	主客同
10	香港马会		0.970	平手/半球	0.800		02-07 01:30	0.870	平手/半球	0.900	02-06 19:27	主客同
11	金宝博		1.140	平手/半球	0.710		02-07 14:00	1.000	平手/半球	0.840	02-06 19:55	主客同
12	12BET (易胜博)		0.800	平手 降	0.950	1 ▼	02-07 06:58	0.950	平手/半球	0.810	02-06 20:05	主客同
13	必发		0.850	平手 降	1.040	10 ▼	02-07 13:56	0.760	平手/半球	0.450	02-06 17:44	主客同
14	BETDAQ		1.120	平手/半球	0.650		02-07 14:26	0.960	平手/半球	0.820	02-06 20:39	主客同
15	Leon		0.850	平手 降	1.040	2 ▼	02-07 14:03	1.010	平手/半球	0.830	02-06 23:33	主客同
16	Betmonsters		0.750	平手	0.950	3 ▼	02-07 13:27	0.820	平手	0.880	02-03 17:38	主客同
17	申博138		0.800	平手 降	0.950	1 ▼	02-07 06:58	0.950	平手/半球	0.810	02-06 20:05	主客同
18	888Sport.es		1.170	平手/半球	0.640	1 ▼	02-07 04:00	0.930	平手/半球	0.810	02-06 20:47	主客同
19	Babibet		0.830	平手 降	0.980	4 ▼	02-07 02:56	1.000	平手/半球	0.840	02-06 19:55	主客同
20	ArtemisBet		0.730	平手 降	0.971	5 ▼	02-07 13:59	0.917	平手/半球	0.769	02-06 20:05	主客同

图 7-18 亚赔网页



- 在欧赔网页、亚赔网页和数据分析网页的顶部,有一个共同的导航条,如图 7-20。



不同彩票网站的比赛数据导航条形式可能不同，不过大体都类似。需要注意的是，各大网站中的投注分析，一般仅指该网站自身的投注数据，有很大的局限性。

7.5.2 案例 7-13: 提取赔率数据

案例 7-13 的文件名是 `zc713_xdat002.py`, 本案例将完成最后的临门一脚, 介绍如何使用函数 `fb_gid_getExt_oz4htm` 从 HTML 网页信息中提取赔率数据。

为了便于学习 `fb_gid_getExt_oz4htm` 函数, 我们省略了网页的下载等过程, 直接读取 `dat` 目录下的赔率数据网页文件进行分析。

同时, 把 `fb_gid_getExt_oz4htm` 函数代码复制到主流程, 便于讲解。

案例 7-13 的主流程代码分成几组, 我们分组进行讲解。

第 1 组代码如下:

```
#1
gid='240228'
fgid='/tfbDat/gid2017.dat'
gids=pd.read_csv(fgid,index_col=False,dtype=str,encoding='gbk')
```

第 1 组代码用于设定参数, 并读取 `gids` 比赛数据文件。

第 2 组代码如下:

```
#2
g10=gids[gids['gid']==gid]

bars=pd.Series(list(g10.values[0]),index=list(g10))
print('\n#2')
print(bars)
print('\ntype(g10)',type(g10))
```

第 2 组代码从 `gids` 内存数据库中找到指定的 `gid` 对应的比赛数据, 保存到变量 `g10` 中。

同时初始化 `bars` 数据节点变量, 最后输出 `g10` 和 `bars` 保存的数据和格式:

```
#2
gid                240228
gset               亚选赛
mplay             叙利亚
mtid              94
gplay            黎巴嫩
gtid              106
qj                 4
qs                 0
```

```

qr                0
kend              1
kwin              3
kwinrq           -1
tweek            3
tplay            2010-03-03
tsell            2010-03-03 20:29
dtype: object

```

```
type(g10), <class 'pandas.core.frame.DataFrame'>
```

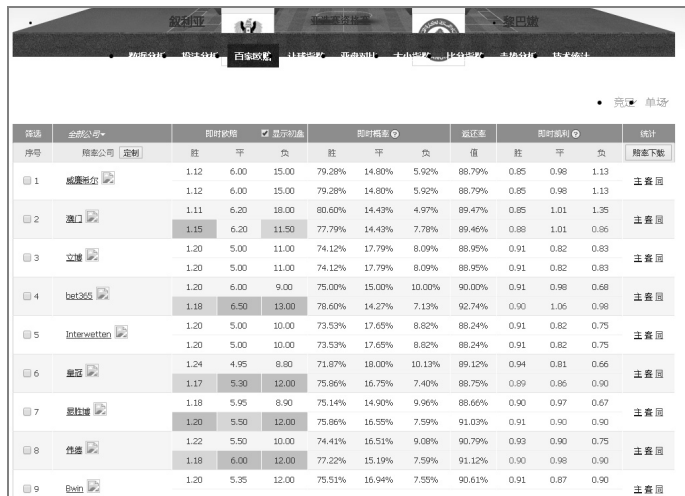
第3组代码如下：

```

#3
fhtm,ftg='dat/'+gid+'_oz.htm','tmp/'+gid+'_xd.dat'
htm=zt.f_rd(fhtm)
df=fb_gid_getExt_oz4htm(htm,bars,ftg)
print('\n#3')
print(df.tail())

```

根据参数读取 HTML 文件，网页文件内容如图 7-21 所示。



序号	赔率公司	定制	胜	平	负	即时胜率	即时赔率	返还率	即时胜率	胜	平	负	统计
1	威廉希尔		1.12	6.00	15.00	79.28%	14.80%	5.92%	88.79%	0.85	0.98	1.13	注意图
2	澳门		1.11	6.20	18.00	80.60%	14.43%	4.97%	89.47%	0.85	1.01	1.35	注意图
3	立博		1.20	5.00	11.00	74.12%	17.79%	8.09%	88.95%	0.91	0.82	0.83	注意图
4	bet365		1.20	6.00	9.00	75.00%	15.00%	10.00%	90.00%	0.91	0.98	0.68	注意图
5	Interwetten		1.20	5.00	10.00	73.53%	17.65%	8.82%	88.24%	0.91	0.82	0.75	注意图
6	皇冠		1.24	4.95	8.80	71.87%	18.00%	10.13%	89.12%	0.94	0.81	0.66	注意图
7	易胜博		1.18	5.95	8.90	75.14%	14.90%	9.96%	88.66%	0.90	0.97	0.67	注意图
8	伟德		1.22	5.50	10.00	74.41%	16.51%	9.08%	90.79%	0.93	0.90	0.75	注意图
9	Bwin		1.18	6.00	12.00	77.22%	15.19%	7.59%	91.12%	0.90	0.98	0.90	注意图

图 7-21 赔率网页截图

如图 7-21 所示的网页因为是离线文件，所以与 500 彩票网站的部分图片有差异，不过不影响数据提取。

在案例中，通过调用函数 `fb_gid_getExt_oz4htm` 提取相关的赔率数据，保存到 `df` 变量，对应的输出信息如下：

```
#3
      gid  cid  cname pwin0 pdraw0 plost0 pwin9 pdraw9 plost9
vwin0  ...  mtid gplay gtid qj qs qr kwin kwinrq tweek  tplay
28 240228 413 Bet3000 1.20 6.00 13.00 1.20 6.00 13.00
77.38 ... 94 黎巴嫩 106 4 0 0 3 -1 3 2010-03-03
29 240228 525 Bet7days 1.23 5.50 10.00 1.23 5.50 10.00
74.26 ... 94 黎巴嫩 106 4 0 0 3 -1 3 2010-03-03
30 240228 90005 gavg 1.19 5.42 11.14 1.18 5.53 12.05
75.07 ... 94 黎巴嫩 106 4 0 0 3 -1 3 2010-03-03
31 240228 90009 gmax 1.25 7.20 18.00 1.23 7.00 18.50
80.60 ... 94 黎巴嫩 106 4 0 0 3 -1 3 2010-03-03
32 240228 90001 gmin 1.11 4.10 5.90 1.11 4.10 8.74
69.46 ... 94 黎巴嫩 106 4 0 0 3 -1 3 2010-03-03
```

由 `df` 的输出数据可以看到，网页文件中的赔率数据已经基本提取完成。

7.5.3 赔率数据与结构化数据

提取网页赔率数据，简单来说，就是把不规范的网页信息转换为标准的结构化数据，便于后期的数据分析。

在案例中，使用的是自定义函数 `fb_gid_getExt_oz4htm`，对应的代码如下：

```
def fb_gid_getExt_oz4htm(htm,bars,ftg=''):
    bs=BeautifulSoup(htm,'html5lib') # 'lxml'
    x10=bs.find_all('tr',ttl='zy')
    df=pd.DataFrame(columns=tfsys.gxdatSgn)
    ds=pd.Series(tfsys.gxdatNil,index=tfsys.gxdatSgn)
    xc,gid=0,bars['gid']
    xlst=['gset','mplay','mtid','gplay','gtid',
'qj','qs','qr','kwin','kwinrq','tplay','tweek']
    for xc,x in enumerate(x10):
        #print('\n%x\n',xc,'#',x.attrs)
        x2=x.find('td',class_='tb_plgs');#print(x2.attrs)
        ds['gid'],ds['cid'],ds['cname']=gid,x['id'],x2['title']
```



```

#
x20=x.find_all('table',class_='pl_table_data');
clst=zt.lst4objs_txt(x20,['\n','\t','%'])
ds=tft.fb_gid_getExt_oz4clst(ds,clst)
#
zdat.df_2ds8xlst(bars,ds,xlst)
df=df.append(ds.T,ignore_index=True)

#
#print('xx',xc)
#--footer
if xc>0:
    x10=bs.find_all('tr',xls='footer')

    for xc,x in enumerate(x10):
        #print('\n@x\n',xc,'#',x.attrs)
        if xc<3:
            x20=x.find_all('table',class_='pl_table_data');
            clst=zt.lst4objs_txt(x20,['\n','\t','%'])
            ds['gid']=gid
            if xc==0:ds['cid'],ds['cname']='90005','gavg'
            if xc==1:ds['cid'],ds['cname']='90009','gmax'
            if xc==2:ds['cid'],ds['cname']='90001','gmin'
            #
            zdat.df_2ds8xlst(bars,ds,xlst)
            ds=tft.fb_gid_getExt_oz4clst(ds,clst)
            #
            df=df.append(ds.T,ignore_index=True)

#
if ftg!='':df.to_csv(ftg,index=False,encoding='gb18030')
#
return df

```

函数 `fb_gid_getExt_oz4htm` 的代码看起来较长，其实很简单，就是两大部分，第一部分提取各行的赔率数据，第二部分提取网页底部的最大值、最小值和平均值等数据。

函数的两组代码结构也大体类似，网页数据的提取使用的还是 `Beautiful Soup`

模块库，这方面的细节读者可以回顾前面 gid 比赛数据的案例，已经讲解得非常详细，这里不再重复介绍。

函数中使用了一个小的技巧，即调用 `zdat.df_2ds8x1st` 函数从 bars 数据包节点获得 ds 的赋值，相关代码如下：

```
xlst=['gset','mplay','mtid','gplay','gtid',
'qj','qs','qr','kwin','kwinrq','tplay','tweek']
zdat.df_2ds8x1st(bars,ds,xlst)
```

函数的两组代码都调用了 `zdat.df_2ds8x1st` 函数，其代码是：

```
def df_2ds8x1st(df,ds,xlst):
    for xss in xlst:
        ds[xss]=df[xss]
    return ds
```

函数 `fb_gid_getExt_oz4htm` 的流程图较长，分为两段，如图 7-22 和图 7-23 所示。

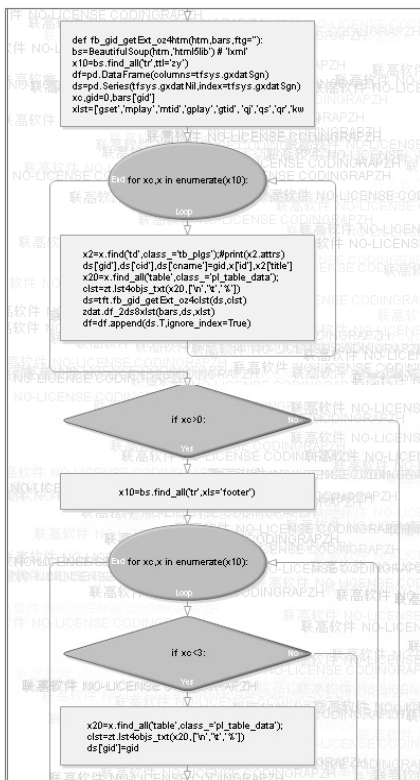


图 7-22 函数流程图 1

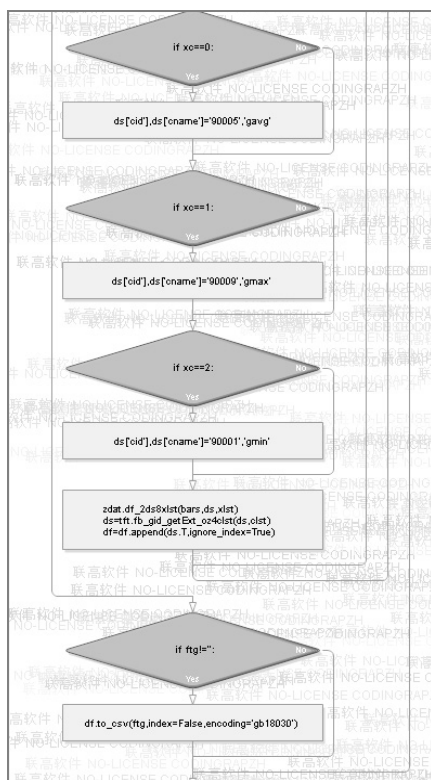


图 7-23 函数流程图 2

7.5.4 瀑布流数据网页与小数据理论

前面说到：
由 df 的输出数据可以看出，网页文件中的赔率数据已经基本提取完成。
之所以说“基本提取完成”，而不是“100%提取完成”，是因为 500 彩票网站虽然是开放的足彩网站，但还是对部分数据采取了保护措施，其中的赔率数据网页和其他彩票网站一样，采用了瀑布流的动态网页技术，默认只提供最主要的 30 家公司的赔率数据。

在案例 7-13 中，使用的比赛 gid 编码是 240228，对应的赔率网址是 <http://odds.500.com/fenxi/ouzhi-240228.shtml>。

如图 7-24 所示是 500 彩票网站赔率网页的底部，如图 7-25 所示是对应的赔率网页离线文件的底部。

<input type="checkbox"/> 252 1xBet		2.63	3.44	2.85	37.21%	28.45%	34.34%	97.87%	1.11	0.99	0.83	主客同	
		2.28	3.46	3.36	42.78%	28.19%	29.03%	97.54%	0.96	0.99	0.98		
<input type="checkbox"/> 253 24hBET		2.60	3.30	2.77	36.68%	28.90%	34.43%	95.36%	1.10	0.95	0.81	主客同	
		2.27	3.30	3.30	42.09%	28.95%	28.95%	95.55%	0.96	0.95	0.96		
<input type="checkbox"/> 254 5Dimes		2.12	3.09	3.09	42.16%	28.92%	28.92%	89.37%	0.90	0.89	0.90	主客同	
		2.30	3.38	3.42	42.50%	28.92%	28.58%	97.75%	0.97	0.97	1.00		
<input type="checkbox"/> 255 888Sport		2.40	3.40	2.85	39.25%	27.70%	33.05%	94.19%	1.01	0.97	0.83	主客同	
		2.16	3.20	3.35	43.11%	29.10%	27.79%	93.11%	0.91	0.92	0.98		
<div>显示选择 反选</div> <div>数据默认 共255家公司</div> <div><input type="checkbox"/> 国家 <input type="checkbox"/> 头尾赔率</div>		平均值	2.35	3.27	2.86	39.37%	28.25%	32.39%	92.18%	0.99	0.94	0.83	下载
			2.20	3.24	3.18	42.22%	28.62%	29.16%	92.72%	0.93	0.93	0.93	
		最高值	2.65	3.50	3.38	43.46%	30.35%	35.48%	98.51%	1.12	1.00	0.98	
		最低值	2.00	2.62	2.22	35.82%	26.62%	28.39%	76.17%	0.84	0.75	0.65	
		1.99	2.90	2.62	36.24%	27.01%	26.85%	83.08%	0.84	0.83	0.76		
		高数值	35.29	43.12	289.8								
			9.9	8.32	19.04								

图 7-24 500 彩票网站赔率网页的底部

<input type="checkbox"/> 27	Bet-at-home		1.20	5.25	10.00	74.15%	16.95%	8.90%	88.98%	0.91	0.86	0.75	主客同
			1.20	5.25	10.00	74.15%	16.95%	8.90%	88.98%	0.91	0.86	0.75	
<input type="checkbox"/> 28	Bet24		1.17	5.60	12.00	76.54%	15.99%	7.46%	89.56%	0.89	0.91	0.90	主客同
			1.17	5.60	12.00	76.54%	15.99%	7.46%	89.56%	0.89	0.91	0.90	
<input type="checkbox"/> 29	Bet3000		1.20	6.00	13.00	77.38%	15.48%	7.14%	92.86%	0.91	0.98	0.98	主客同
			1.20	6.00	13.00	77.38%	15.48%	7.14%	92.86%	0.91	0.98	0.98	
<input type="checkbox"/> 30	Bet7days		1.23	5.50	10.00	74.26%	16.61%	9.13%	91.34%	0.94	0.90	0.75	主客同
			1.23	5.50	10.00	74.26%	16.61%	9.13%	91.34%	0.94	0.90	0.75	
<div><div><div>显示选择</div><div>反选</div></div></div>		平均值	1.19	5.42	11.14	75.07%	16.65%	8.28%	89.58%	0.91	0.88	0.84	下载
		最高值	1.18	5.53	12.05	75.93%	16.40%	7.66%	89.87%	0.90	0.90	0.91	
数据默认 共114家公司		最低值	1.25	7.20	18.00	80.60%	21.69%	13.60%	93.82%	0.95	1.17	1.35	
			1.23	7.00	18.50	80.89%	21.21%	10.07%	97.06%	0.94	1.14	1.39	
<div><div><div><input type="checkbox"/> 国家</div><div><input type="checkbox"/> 头尾赔率</div></div></div>		最低值	1.11	4.10	5.90	69.46%	12.90%	4.95%	80.21%	0.85	0.67	0.44	
		高数值	1.11	4.10	8.74	71.79%	13.49%	4.95%	83.56%	0.85	0.67	0.66	
		高数值	4.93	76.47	292.98								
			3.47	87.56	271.84								

图 7-25 赔率网页离线文件的底部

图 7-24 中左边的博彩机构序号是 255，表示这场比赛 500 彩票网站有 255 家不同的赔率机构开出的赔率数据，这个数字不是固定的，和具体的比赛有关，有些比赛的数字高，有些比赛的数字低。

图 7-25 中左边的博彩机构序号是 30，表示离线文件只保存了 30 家主要博彩机构的赔率数据。

虽然 500 彩票网站采用了瀑布流的动态网页技术，但抓取全部数据的技术方法也很简单，例如模拟键盘鼠标，不断拖动鼠标到浏览器网页的底部，待全部网页显示完毕再进行抓取，就是时间慢一点，每个网页需要数秒，而且无法多开抓取，除非采用虚拟机等其他技术配合。

tfbDat 足彩数据包的早期版本是 zcDat，也曾经抓取过全部的赔率数据，不过实盘分析时发现，完全没有必要使用这么多机构的数据。

这里面涉及到一些很基础的算法理论和编程哲学，非常复杂，读者可以参考笔者博客小数据理论相关的文档，本书后面的案例也会部分涉及相关的环节。

8

第 8 章

足彩数据回溯测试

回溯测试是金融量化的名称，是采用历史数据对数据模型进行测试，从而检验模型的好坏。

Top Football 极宽足彩量化分析系统，又称 Top Quant for Football，简称 TFB，是 Top 极宽量化开源团队根据 TopQuant 极宽量化系统，专门针对足彩行业，进行移植的版本，这可能是足彩领域首个专业的量化回溯系统。

TFB 是自主开发的、纯 Python 语言足彩量化分析系统，基于 TopQuant（原名 zwQuant）极宽量化策略分析及回溯测试系统，内置人工智能、机器学习模块，提供策略分析和回溯测试功能，是一套全功能的、全开源的、纯 Python 量化策略分析系统，可以直接用于足彩实盘操作。

在数据源方面，配合 tfbDat 和内置的足彩数据更新程序，用户可以完成足彩比赛数据、赔率数据的网络采集、数据清洗、自动去重和数据更新。

前面已经介绍过 TFB 足彩量化程序的系统架构和 tfbDat 足彩数据包的结构，本章继续采用逆向工程模式，首先通过具体的案例介绍 TFB 足彩量化程序的实盘数据更新、赔率分析，然后由上至下慢慢讲解各个模块的相关功能。

Top Football 极宽足彩量化分析系统正在不断开发和完善当中，和其他各种开源项目一样，未来版本收录的模块和源码也会不断发生变化，如果本书内容和具体源码有冲突，通常以最新发布的软件源码为准。

为了方便读者尽快熟悉 TFB 系统和配套的 tfbDat 足彩数据包，下面先介绍 TFB

系统的基本结构。

8.1 TFB系统构成

8.1.1 TFB 系统模块结构

TFB 极宽足彩量化分析系统 source 目录下，虽然文件繁多，但 TFB 系统构成非常简单，如图 8-1 所示。



图 8-1 TFB 系统构成

由图 8-1 可以看出，TFB 主要由以下三大模块构成：

- Top-Base，极宽基础模块库；
- Top-Football，极宽足彩专业模块库；
- tfbDat，极宽足彩数据包。

8.1.2 Top-Base 极宽基础模块库

Top-Base 极宽基础模块库是极宽各种系统的基础模块库，各个模块文件均采用 z 字母作为文件首字母，一方面为了和各个应用模块区别，另外一方面表示极宽系统源自最初的 zw 量化系统。

如图 8-2 所示是 Top-Base 极宽基础模块库的主要模块组成。



图 8-2 Top-Base 极宽基础模块库主要模块组成

Top-Base 极宽基础模块库包括以下模块：

- zsys，全局系统模块，用于定义全局参数、变量、常数等；
- ztools，常用工具函数库，缩写为 zt；
- ztools_str，常用字符串函数库，缩写为 zstr；
- ztools_web，常用 Web 网页抓取、HTML 分析函数库，缩写为 zweb；
- zdraw，常用绘图函数库；
- ztop_data，极宽数据分析、预处理函数库，缩写为 zdat；
- ztop_ai，极宽通用机器学习模块库，缩写为 zai。

需要说明的是，未来随着极宽各个应用软件和系统的升级和发展，各个应用模块的构成可能会有所变动，请读者注意。

此外，因为字符串处理和 Web 网页数据抓取分析使用频率较多，所以从原来的 ztools 常用工具函数库分离出来。

8.1.3 Top-Football 极宽足彩专业模块库

Top-Football 极宽足彩专业模块库是专门针对足彩行业开发的函数库，各个模块文件均采用 tfb 字母作为文件开头。如图 8-3 所示是 Top-Football 极宽足彩专业模块库的主要模块组成。

Top-Football 极宽足彩专业模块库包括以下模块：

- tfb_main，足彩主程序入口模块；
- tfb_sys，足彩系统模块库，足彩系统所需的类定义、全局参数和变量，缩写为 tfsys；



图 8-3 Top-Football 极宽足彩模块库主要模块组成

- tfb_tools, 足彩系统常用工具函数库, 缩写为 tft;
- tfb_draw, 足彩系统常用绘图函数库, 缩写为 tfdr;
- tfb_data, 足彩系统数据分析、预处理函数库, 缩写为 tfdat;
- tfb_strategy, 足彩系统常用策略模块库, 缩写为 tfsty;
- tfb_backtest, 足彩常用回溯分析函数库, 缩写为 tfbt。

需要说明的是, 未来随着极宽各个应用软件和系统的升级和发展, 各个应用模块的构成可能会有所变动, 请读者注意。

8.2 实盘数据更新

8.2.1 案例 8-1: 实盘数据更新

案例 8-1 其实是源自 TFB 系统的实际工作程序, 原来的文件名是 main_get.py 足彩数据更新程序, 案例 8-1 的文件名是 zc801_main_get.py, 只是在原来的文件名前加了教学课件的学号前缀“zc801_”。

案例 8-1 介绍在 TFB 系统中更新足彩数据, 包括下载 gid 比赛数据网页、下载赔率数据网页、提取 gid 比赛数据和赔率数据。

案例 8-1 主程序很短, 代码如下:

```
main_get('', 2)
print('\nok!')
```

调用 main_get 函数更新足彩数据, 调用 main_get 函数定义代码如下:

```
def main_get(timStr='', nday=2):
```


参数说明如下：

- **timStr**，回溯结束日期，yyyy-mm-dd 格式，回溯采用倒退计算模式，所以输入的是结束的日期；
 - **nday**，表示需要下载的数据天数，默认 **nday=2**，下载最近两天的数据。
- 调用时，参数 **nday** 有两个特殊参数：
- 如果 **nday=0**，则调用后不会下载数据，而是输出相关的 **gid** 信息。
 - 如果 **nday=-1**，则实际下载数据的天数是系统默认足彩数据起始时间到运行当日之间的天数再加上 10。

很久没有更新足彩数据或者刚刚下载 **tfbDat** 足彩数据，第一次运行 **main_get.py** 足彩数据更新程序时，可以使用参数 **nday=0** 调用程序：

```
main_get('',0)
```

对应的输出信息如下：

```
main_get,nday: 0
now: 2017-02-10
gid tim0: 2010-01-01, nday: 2597
gid tim9: 2017-02-06, nday: 4
```

输出信息中的 **tim0** 信息是：

```
gid tim0: 2010-01-01, nday: 2597
```

以上代码表示 TFB 足彩系统数据起始时间是 2010-01-01，距程序运行当天已经有 2597 天。如果使用 **nday=-1**，则调用 **main_get** 函数，实际的 **nday=2597+10**，即 2607 天，读者不要担心这个日期会超出数据起始时间，系统会自动忽略以前的日期，不进行处理。

8.2.2 实盘要点：冗余

输出信息中的 **tim9** 信息是：

```
gid tim9: 2017-02-06, nday: 4
```

以上代码是 **gid** 数据中最后一条比赛数据的比赛日期，其中，**nday** 表示距程序运行当天的天数，通常在实际更新数据时，需要在 **nday** 数值上加 2~3 天，以表示冗余。

足彩数据与股票不同，采用的是预售模式，通常会预售 2~3 天的比赛，在世界杯期间，更会提前半个月预售 8 强、16 强、冠亚军的足球彩票，而赔率数据又是动

态变化的，不到比赛开始，无法得到最后的收盘赔率数据。

以刚才的输出信息为例：

- 最后的 `tim9` 信息显示的是 2 月 6 日。
- 实际上，这条数据应该是在 2 月 4~5 日抓取的，甚至更早。
- 笔者测试的日期是 2 月 10 日，如果取 `nday=4`，从 2 月 6 日开始抓取，则 4~5 日的的数据不是 24 点后抓取的，是当天的收盘赔率数据，并非最终的收盘赔率数据，会引起系统混乱。
- 实盘需要设置 `nday=6` 或者 `nday=7`，冗余几天，覆盖 2 月 4~5 日的日期，抓取最终的比赛数据，并更新 `tfbDat` 数据包的数据。

这部分内容读者可以多体会一下，通过运行程序，实际查看 `gid` 比赛数据文件 `gid2017` 和相关的赔率数据文件。

有趣的是，笔者更新完毕后，在取 `nday=0` 时，调用 `main_get` 函数，对应的输出信息如下，其中 `nday` 是负数：

```
main_get,nday: 0
now: 2017-02-10T19:22:30.609942+08:00
gid tim0: 2010-01-01, nday: 2597
gid tim9: 2017-02-13, nday: -3
```

笔者测试时是 2 月 10 日，上面的 `tim9` 最后一条数据的最终比赛日期是 2 月 13 日，是未来的 3 日后，所以 `nday=-3`。

知道足彩采用预售机制，可以提前销售未来数日的比赛彩票，读者就能明白其中的道理了。

8.2.3 实盘要点：耐心

下载 `gid` 比赛数据，只要网速正常，的确立马可得，可是每个 `gid` 比赛网页不过是众多 `gid` 比赛编码的索引文件，前面我们曾经计算过，平均每天有 20~30 场足球比赛，周末更是多达上百场，这样每次下载更新，即使 2 天，也有 50~100 个网页需要下载。

`main_get` 函数带有计时功能，笔者实盘测试，当 `nday=5` 时，下载 5 天的数据，时间如下：

```
#4,update.data,tim:1440.25 s
```

下载 5 天的数据耗时 1440 秒，合 24 分钟，平均下载一天的数据需要 5 分钟。

每次更新，至少需要下载 2 天的数据，大约是 10~20 分钟，所以读者要有耐心，同时注意将时间提前，不要超过最终的投注时间，耽误实盘下单。

8.2.4 实盘要点：数据文件

每次运行更新后的程序，系统都会自动修改 gid 数据文件，以及保存各种相关的网页和数据文件，默认数据和目录位置如下。

- tfbDat 数据包默认目录：\tfbDat\。
- gid 比赛基本数据文件：\tfbDat\gid2017.dat。
- rghrm, gid 网页保存目录，默认路径是\tfbDat\xhtm\ghrm\。
- rxhtm, 赔率网页保存目录，默认路径是\tfbDat\xhtm\。
- rhtmOuzhi, 欧赔网页保存目录，默认路径是\tfbDat\xhtm\htm_oz\。
- xdat, 提取后的欧赔数据保存目录，默认路径是\tfbDat\xdat\。

读者可以在运行前先保存 gid 比赛基本数据文件\tfbDat\gid2017.dat，做一下备份，运行完毕后看看两个文件尾部的数据有什么不同。

同时，按最新时间排序，看看以上目录中增加的新文件。

8.2.5 main_get 函数

本节介绍 main_get 函数，main_get 函数的代码不长，分为几组，下面逐一进行介绍。

main_get 函数第 1 组代码如下：

```
#1---init.sys
print('\nmain_get,nday:',nday)
tfbsys.xnday_down=nday
zsys.web_get001txtFg=True
```

第 1 组代码用于设置两个系统全局变量：

- tfbsys.xnday_down, 数据下载的天数；
- zsys.web_get001txtFg, 是否强制下载网页文件。

zsys.web_get001txtFg=True 表示强制下载所有网页文件，即使本地硬盘已经有

文件名对应的文件。如果读者想自己下载所有网页数据，第 1~2 轮下载时可以设置为 True，采用强制下载模式，后面几轮属于补漏性质，参数可以设置为 False，只下载本地没有的网页。

在数据追加更新时，要将参数设置为 True，强制下载所有网页数据。

main_get 函数第 2 组代码是：

```
#2---init.tfb
rs0='/tfbDat/'
fgid=rs0+'gid2017.dat'
xtfb=tft.fb_init(rs0,fgid)
if nday== -1:
    tfsys.xnday_down=xtfb.gid_nday+10
    print('nday,',tfsys.xnday_down)
```

以上代码用于设置 gid 文件名、tfbDat 数据包目录，调用 tft.fb_init 函数，对变量 xtfb 进行初始化设置。

此外，如果调用参数 nday=-1，则重新设置 nday 为下载全部数据模式。

main_get 函数第 3 组代码是：

```
#3---update.data
print('\n#3,update.data')
if nday!=0:
    tft.fb_gid_get_nday(xtfb, timStr,fgExt=True)
```

第 3 组代码是 main_get 函数的真正核心所在：

- 调用 tft.fb_gid_get_nday 函数抓取数据；
- 设置参数 fgExt=True，表示抓取赔率等扩展数据。

main_get 函数第 4 组代码是：

```
#4
tn=zt.timNSec('',xtfb.tim0,'')
print('\n#4,update.data,tim:{0:.2f} s'.format(tn))
#
```

运行完毕，输出计时数据。

main_get 函数的核心代码是：

```
tft.fb_gid_get_nday(xtfb, timStr,fgExt=True)
```

这行代码调用 tft.fb_gid_get_nday 函数抓取数据，该函数已经在前面的章节具体介绍过。此外，还需要注意下面的代码：

```
xtfb=tft.fb_init(rs0,fgid)
```

用于 `xtfb` 变量进行初始化。

8.3 变量初始化

在 TFB 足彩量化系统和 TopQuant 金融量化系统中，程序开头都有类似的代码：

```
xtfb=tft.fb_init(rs0,fgid)
```

用于对 `xtfb` 变量进行初始化设置。

`tft.fb_init` 足彩初始化函数位于 `tfb_tools` 模块包，包括多组代码，下面逐一进行讲解。

`tft.fb_init` 足彩初始化函数第 1 组代码如下：

```
#1
xtfb=tfsys.zTopFoolball()
xtfb.tim_now=arrow.now()
xtfb.timStr_now=xtfb.tim_now.format('YYYY-MM-DD')
xtfb.tim0,xtfb.tim0Str=xtfb.tim_now,xtfb.timStr_now
print('now:', zt.tim_now_str())
```

这段代码从 `tfsys` 的 `zTopFoolball` 类定义，衍生出 `xtfb` 变量，稍后细说。

设置备份时间参数，并显示运行时的实际时间。

时间显示格式设置很容易搞错，这里有一个小技巧，使用极宽常用工具函数模块的 `zt.tim_now_str()` 函数来替换当前的时间字符串数据，相关的函数代码如下：

```
def tim_now_str():
    dss=arrow.now().format('YYYY-MM-DD HH:mm:ss')
    return dss
```

`tft.fb_init` 足彩初始化函数第 2 组代码如下：

```
#2
#xtfb.pools=[]
xtfb.kcid='1' #官方,3=Bet365
xtfb.funPre=tfsty.sta00_pre
xtfb.funSta=tfsty.sta00_sta
```

这段代码设置 `xtfb` 变量的部分参数，其中 `kcid` 是最终结算的博彩机构，默认是我国官方机构，其他的还有 `Bet365` 等。

`funPre` 是通用数据预处理函数，`funSta` 是通用策略函数，这两个 `sta00` 函数都是空函数。

这种类似变量赋值的方法是 Python 语言强大而又优雅的一个特征，其他编程语言如 Delphi、C、Java 要实现类似功能，需要复杂、烦琐的指针、包装等，要用几百行代码，而 Python 只需一行代码。

tft.fb_init 足彩初始化函数第 3 组代码如下：

```
#3
if rs0!='':
    tfsys.rdat=rs0
    tfsys.rxdat=rs0+'xdat/'
    tfsys.rhtmOuzhi=rs0+'xhtm/htm_oz/'
    tfsys.rhtmYazhi=rs0+'xhtm/htm_az/'
    tfsys.rhtmShuju=rs0+'xhtm/htm_sj/'
```

这段代码表示如果 tfbDat 数据包文件目录有变化，则通过 rs0 参数可以重新配置相关的数据目录。

tft.fb_init 足彩初始化函数第 4 组代码如下：

```
#4
if fgid!='':
    tfsys.gidsFN=fgid
    #xtfb.gids=pd.read_csv(fgid,index_col=0,dtype=str,encoding='gbk')
    tfsys.gids=pd.read_csv(fgid,index_col=False,dtype=str,
encoding='gbk')
    fb_df_type_xed(tfsys.gids)
    tfsys.gidsNum=len(tfsys.gids.index)
    #-----tim.xxx
    xtfb.gid_tim0str,xtfb.gid_tim9str=tfsys.gids['tplay'].min(),
tfsys.gids['tplay'].max()
    tim0,tim9=arrow.get(xtfb.gid_tim0str),arrow.get(xtfb.gid_tim9str)
    xtfb.gid_nday,xtfb.gid_nday_tim9=zt.timNDay('',tim0),zt.timNDay
('',tim9)
    print('gid tim0: {0}, nday: {1}'.format(xtfb.gid_tim0str,
xtfb.gid_nday))
    print('gid tim9: {0}, nday: {1}'.format(xtfb.gid_tim9str,
xtfb.gid_nday_tim9))
```

第 4 组代码的功能分为两部分，前提都是 gid 文件名存在，首先是调用部分，读取 gid 数据；然后根据读取的 gid 数据计算相关数据，主要是时间数据。

8.3.1 全局变量与类定义

tft.fb_init 初始化函数中使用了以下语句：

```
xtfb=tfsys.zTopFoolball()
```

从 tfsys 的 zTopFoolball 类定义，衍生出 xtfb 变量，笔者曾经讲过，尽量不要采用面向对象的模式来编写 Python 程序。

这里也是如此，虽然 xtfb 源自 zTopFoolball 类定义，但这里的类只有一个数据容器，类似 Delphi 的 record 纪录和 C 语言的 struct 结构数据，是一个单纯的数据包，没有任何方法定义和继承之类的语法。

使用 xtfb 这样一体化的变量包作为通用变量，在系统的各个函数间交换数据，可以简化函数设计、统一格式、扩展方便。

日后如果需要增加某个参数、变量，无需修改各个函数的接口，在类定义方面进行调整即可。

zTopFoolball 类定义位于 tfb_sys.py 极宽足彩量化软件系统参数模块，相关源码如下：

```
class zTopFoolball(object):
    '''
    设置 TopFoolball 项目的各个全局参数
    尽量做到 all in one
    '''
    def __init__(self):
        #----rss.dir
        self.tim0Str_gid='2010-01-01'
        self.tim0_gid=arrow.get(self.tim0Str_gid)

        #
        self.gid_tim0str,self.gid_tim9str='', ''
        self.gid_nday,self.gid_nday_tim9=0,0
        #
        self.tim0,self.tim9,self.tim_now=None,None,None
        self.tim0Str,self.tim9Str,self.timStr_now='',' ',' '
        #

        self.kgid=''
        self.kcid=''
```

```

self.ktimStr=''
#
#---pool.1day
self.poolInx=[]
self.poolDay=pd.DataFrame(columns=poolSgn)
#---pool.all
self.poolTrd=pd.DataFrame(columns=poolSgn)
self.poolRet=pd.DataFrame(columns=retSgn)

self.bars=None
self.gid10=None
self.xdat10=None

#
#--backtest.var
self.funPre,self.funSta=None,None
self.preVars,self.staVars=[],[]

#
#--ret.var
self.ret_nday,self.ret_nWin=0,0
self.ret_nplay,self.ret_nplayWin=0,0

self.ret_msum=0

```

因为 zTopFootball 类定义还在不断优化调整中，所以以上代码仅供参考。

需要说明的是，TFB 足彩量化分析系统还在不断开发完善中，tft.fb_init 初始化函数也在不断优化中，如果有所变化，请以最新发布的软件代码为准。

此外，为了方便讲解和使用，案例 8-1 中的 main_get 函数是放在主程序中的，实际更新程序也是，不过在 tfb_main 主程序入口模块中，也有一个完全一样的 main_get 函数备份。

8.3.2 彩票池内存数据库

在 TFB 足彩量化分析系统中，还借用了量化中的股票池的概念来保存多个足彩的数据，在 TFB 系统中称为彩票池。

在 `tfb_sys.py` 极宽足彩量化软件系统参数模块的 `zTopFootball` 类定义中，对应的相关代码如下：

```
self.poolInx=[]
self.poolDay=pd.DataFrame(columns=poolSgn)
#pool.all
self.poolTrd=pd.DataFrame(columns=poolSgn)
self.poolRet=pd.DataFrame(columns=retSgn)
```

在 TFB 系统中，彩票池包括：

- `poolInx`，比赛索引数据，只有 `gid` 编码和 1 天的数据；
- `poolDay`，各场比赛的赔率数据，只有 1 天的数据；
- `poolTrd`，总的交易数据，包括多日的的数据，是 `gid` 数据的增强版本，增加了指定的收盘赔率数据；
- `poolRet`，每天的回报率记录，包括多日的的数据。

此外，还在 `tfb_sys.py` 模块中定义了两个全局变量：

```
gids=pd.DataFrame(columns=gidSgn,dtype=str)
xdats=pd.DataFrame(columns=gxdatsSgn,dtype=str)
```

这两个变量其实也属于彩票池范围，原本命名为 `gids_pool` 和 `xdats_pool`，后来为了简化，变量名改为 `gids` 和 `xdats`，分别代表 `gid` 比赛数据和 `xdats` 赔率数据。

变量 `gid` 和 `xdats` 都是采用 Pandas 的 `DataFrame` 数据格式，这样就无需使用 `MySQL` 等数据库系统，大大简化了程序结构。

变量 `gid` 和 `xdats` 直接采用 Pandas 的 `DataFrame` 数据格式，中间无须进行格式转换，在效率上不会低于数据库，但在未来实盘测试时，如果数据量大，影响整体效率了，就需要再做优化处理。

注意：

理论上，32 位 Python 单个字典变量的数据上限是 2GB。64 位版本的 Python 已经不受此限制，数据的上限几乎近于无限。

一线数据分析工程表明，十亿级别以下的数据量都可以用 Pandas 存储并直接分析，效率比 Spark、Hadoop 更高，而且极其方便。

变量 `gid` 和 `xdats` 的数据全部位于内存，可以看成一个 Pandas 版本的内存数据库。

如果采用极宽 GPU 工作站和全内存运算模式，即使不用 GPU 加速，64GB 以上的用户，即使加载 `zwDat` 股票数据源的历年股票数据（约 6GB），也可以全部导入

内存，进行全内存计算。

内存的速度比 SSD 要快 50~100 倍，比硬盘快近千倍，对于 IO 密集型的数据分析项目，整体效率可以提高 10~20 倍，一般的数据分析项目也可以提高 3~5 倍。

读者可参考下面两篇文章：

《zw·10 倍速大数据与全内存计算》：<http://ziwang.com/?p=186>。

《zw 开源量化 GPU 超算工作站草案》：<http://ziwang.com/?p=417>。

8.3.3 案例 8-2：内存数据库&数据包

变量 `gid` 和 `xdata` 的数据全部位于内存，可以看成是一个 Pandas 版本的内存数据库。

有些量化系统把类似的变量称为股票池，如 `stock-pool`，类似传统 IT 编程的线程池，在 TFB 足彩量化系统中称为彩票池。

不管名称如何，目的都是把相关的数据集中处理，提高工作效率。传统的股票池、线程池，一般采用静态镜像模式，池中的数据一般是不变的。

TFB 足彩量化系统的 `gid` 和 `xdata` 变量与传统的股票池、线程池技术略有不同。为了提高效率，采用动态处理的方式，在 `dataPre` 数据预处理阶段，会根据策略要求对彩票池中的数据进行动态衍生扩展。

案例 8-2 很简单，前面也介绍过类似的案例，但是没有结合彩票池的概念，角度完全不同，下面以具体的案例来介绍内存数据库模式的 `gid` 和 `xdata` 变量。

案例 8-2 的文件名是 `zc802_xdat.py`，程序很简单，核心代码如下：

```
#1
rs0='/tfbDat/'
fgid,fxdat=rs0+'gid2017.dat',rs0+'xdat2017.dat'

#2
tim0=arrow.now()
gids=pd.read_csv(fgid,index_col=False,dtype=str,encoding='gbk')
tn=zt.timNSec(' ',tim0)
dn=len(gids.index)
print('#2,gids tim: {0}s,data num:{1:,} '.format(tn,dn))
```

```
#3
tim0=arrow.now()
xdats=pd.read_csv(fxdats,index_col=False,dtype=str,encoding='gb18030')
tn=zt.timNSec(' ',tim0)
dn=len(xdats.index)
print('#3,xdats tim: {0}s,data num:{1:}, '.format(tn,dn))
```

程序分为3组：

- 第1组，设置相关参数；
- 第2组，读取 fgid 文件到 gid 变量；
- 第3组，读取 fxdat 文件到 xdats 变量。

运行结果如下：

```
#2,gids tim: 0.25s,data num:68,711
#3,xdats tim: 15.13s,data num:2,290,966
```

由结果可见，近7万场 gid 比赛数据读入才0.25秒，而 xdats 赔率数据将近230万条，读入也不过15秒，效率完全可以用于实盘分析。

在 TFB 系统中，这种内存数据库是在系统初始化阶段一次性读入的，后面可以反复重新使用。

8.4 回溯测试

回溯测试（back testing）与压力测试（stress testing）是金融工程常用的两个术语，也是经常容易混淆的两个术语。

在金融工程的风控领域，也就是进行风险管理的时候，经常需要进行一些测试，即回溯测试和压力测试。

回溯测试就是用历史数据，对投资策略、资产组合进行测试。

压力测试类似数学的奇点测试，用历史数据中的极端值对投资策略、资产组合进行测试。例如，外汇、美股、海外资金盘可以用美国“1987 股灾”、“9·11”期间的数据；我国股票市场，可以用2008年1月22日A股5000点前后、2015年“七七股灾”和2016年1月熔断期间的数据。这些时期的数据都属于历史上的极值点，可以用于压力测试。

不过，从2015年以来，黑天鹅事件不断，成为新常态，建议进行压力测试时，可以在压力的历史数据上面再提升20%~30%，以强化模型的抗风险能力。

本节开头说了这么多金融工程的事情，是因为 TFB 足彩量化分析系统本身就是源自 TopQuant 极宽金融量化分析系统，是金融工程在足彩领域的具体应用，一些基本的金融背景知识还是需要了解的。

8.4.1 案例 8-3：回溯

案例 8-3 的文件名是 `zc803_main_bt.py`，文件名中的“bt”是 back testing（回溯测试）的英文缩写。

案例 8-3 的主流程和案例 8-2 类似，也是以下代码：

```
main_bt('',0)
print('\nok!')
```

调用 `main_bt` 回溯测试主程序，注意 `main_bt` 的调用参数 `nday=0`，本节只说明 `main_bt` 主体函数，其他细节后面的章节再进行介绍。

`main_bt` 回溯测试函数的定义与 `main_get` 数据更新下载函数的定义一样，都只有一个 `nday` 参数，表示需要运行的天数：

```
def main_bt(timStr='',nday=2):
```

参数说明如下。

- `timStr`，回溯结束日期，使用 `yyyy-mm-dd` 格式，回溯采用倒退计算模式，所以输入的是结束的日期。
- `nday`，表示需要下载的数据天数，默认 `nday=2`，下载最近两天的数据。

调用时，有两个特殊参数：

- 如果 `nday=0`，调用后不会实际回溯数据，而是输出相关的 `gid` 信息；
- 如果 `nday=-1`，`nday` 为实际回溯的天数，是系统默认的足彩数据实际时间到运行当日之间的天数再加上 10。

第一次进行回溯分析，使用参数 `nday=0` 调用程序：

```
main_bt(0)
```

对应的输出信息如下：

```
main_bt,nday: 0
now: 2017-02-11 10:31:58
gid tim0: 2010-01-01, nday: 2598
gid tim9: 2017-02-13, nday: -2
```

```
#3,backtest

#5,backtest,tim:0.31 s

#6,end.main
```

因为调用参数 `nday=0`，没有实际运行回溯程序和回溯结果分析程序，所以输出信息里面缺了第 4 组结果分析的信息：

```
#4,result.anz
```

需要说明的是，在 TFB 足彩量化系统中，历史数据回溯测试与实盘足彩预测函数是一体的，都是 `main_bt` 函数。

默认 `nday=2` 表示处于预测模式，因为实盘更新后，会下载未来数日的数据，例如笔者运行测试的时间是 2 月 11 日，可是其中的 `gid` 数据中最后一条数据是 2 月 13 日，运行 `main_bt` 回溯数据，分析过去两天和未来数日的的数据，其中过去的两天是复核过去的的数据，未来几天的数据属于预测分析了。

如果要进行模型测试，通常 `nday` 需要设置为：

```
100,200,300,500,1000,2000,3000
```

从刚才输出的信息中可以看出，因为 `tim0` 距程序运行当天有 2598 天，设置 3000 天足够未来一年的冗余了，2018 年以后的数据用户可以自己调整。

设置 `nday=3000`，相当于是采用全部数据进行回溯测试。

8.4.2 main_bt 回溯主入口

`main_bt` 回溯主入口函数的代码很简单，已经进行分组，下面逐一进行介绍。

`main_bt` 回溯主入口函数的第 1 组、第 2 组代码如下：

```
#1---init.sys
print('\nmain_bt,nday:',nday)
tfsys.xnday_down=nday
zsys.web_get001txtFg=True

#2---init.tfb
rs0='/tfbDat/'
fgid=rs0+'gid2017.dat'
```

```
xtfb=tft.fb_init(rs0,fgid)
if nday== -1:
    tfsys.xnday_down=xtfb.gid_nday+10
    print('nday,',tfsys.xnday_down)
```

以上代码与 `main_get` 数据更新函数类似，都是设置参数和 `xtfb` 变量，不再重复。`main_bt` 函数的第 3 组代码如下：

```
#
#3---backtest
print('\n#3,backtest')
if nday!=0:
    xtfb.funPre=tfsty.sta00_pre
    xtfb.funSta=tfsty.sta00_sta
    xtfb.preVars=[]
    xtfb.staVars=[]
    xtfb.kcid='1' #cn,3=bet365
#
tfbt.bt_main(xtfb, timStr)
```

第 3 组代码是回溯测试入口，调用 `tfbt.bt_main` 回溯函数，其定义是：

```
def bt_main(xtfb,timStr):
```

其中的 `timStr` 是回溯结束的日期，默认是空字符串，表示是当天的日期。

前面我们讲过，在 TFB 足彩量化系统中，历史数据回溯测试与实盘足彩预测函数是一体的，都是通过 `main_bt` 函数调用 `tfbt.bt_main` 回溯函数。

因为是模型测试，所以建模数据和验证数据应该分开。实盘测试默认使用空字符串，用当天的日期；建模时，如果也是这样，测试数据当中会包含验证数据，影响模型的准确性，所以在实际操作时，一般会预留 3 个月、6 个月或者 1 年的最新数据作为验证数据，在建模时，`timStr` 设置为数月或者一年前的日期，例如“2016-01-01”等。

在调用 `tfbt.bt_main` 函数前，需要进行一些基本的参数设置。

- `xtfb.funPre=tfsty.sta00_pre`，数据预处理函数，这里设置的 `sta00_pre` 函数是空函数，不做任何处理。
- `xtfb.funSta=tfsty.sta00_sta`，策略函数，这里设置的 `sta00_sta` 函数是空函数，不做任何处理。
- `xtfb.preVars=[]`，数据预处理函数配套的参数，不同的函数其参数不同。

- `xtfb.staVars=[]`，策略函数配套的参数，不同的函数其参数不同。
- `xtfb.kcid='1'`，结算时采用的博彩机构编码，默认是中国体彩中心，常用的还有 `bet365`（代码 3）等机构的数据。

`main_bt` 函数的第 4 组代码如下：

```
#
#4---main_ret
print('\n#4,result.anz')
tfbt.bt_main_ret(xtfb,True)
print('kcid',xtfb.kcid)
#
```

调用 `tfbt.bt_main_ret(xtfb,True)` 函数，分析回溯结果并保存相关的数据。

`main_bt` 函数的第 5 组代码如下：

```
#5
tn=zt.timNSec('',xtfb.tim0,'')
print('\n#5,backtest,tim:{0:.2f} s'.format(tn))
```

这段代码也与 `main_get` 数据更新函数类似，计算运行时间并输出相关信息。

为了方便读者使用与移植，TFB 极宽足彩系统中特意增加了一个 `tfb_main` 主模块，定义了一个 `main()` 主函数入口，类似 C 语言，程序入口一目了然。

请注意，`main_bt` 函数和 `tfbt.bt_main` 回溯函数是两个不同的函数，不要搞混。

- `main_bt` 是 `tfb_main` 主模块的回溯测试入口函数，本身不真正执行回溯操作，需要调用 `tfbt.bt_main` 回溯函数完成回溯分析。
- `bt_main` 回溯函数位于 `tfb_backtest` 回溯模块，是具体的回溯函数。

本节导入了两个全新的模块库：

- `tfb_main` 主模块；
- `tfb_backtest` 回溯模块，缩写是 `tfbt`。

8.4.3 案例 8-4：实盘回溯

到目前为止，我们讲解的回溯分析都还是理论上的，下面介绍真正的实盘回溯操作流程。

案例 8-4 的文件名是 `zc804_bt_sta01.py`，是基于实盘的回溯程序，核心代码如下：

```
timStr='2017-02-10'
```

```
tfb_main.main_bt(timStr,5)
print('\nok!')
```

案例 8-4 结构很简单，核心代码就一句：

```
tfb_main.main_bt(timStr,5)
```

以上代码表示调用 `main_bt` 入口函数。

为了保持数据统一，便于讲解，案例 8-4 设置了回溯结束日期：

```
timStr='2017-02-10'
```

如果 `timStr` 为空，则从当前日期开始回溯，既是足彩量化回溯分析，也是足彩推荐预测分析，这句话读者好好理解一下。

使用 `main_bt` 主入口函数进行实盘分析时，需要设置许多参数，特别是不同的策略，需要对应不同的数据预处理函数、策略函数和不同的参数列表，因此 `main_bt` 主入口函数采用的是模板的形式，在实盘分析时，要把函数代码复制到主流程再进行调用。而 TFB 极宽量化分析系统的其他模块库和其他函数都是直接调用。

这样做的另外一个好处是，可以增加一些额外的代码，进行更多的定制，案例 8-4 就在函数尾部增加了几行参数输出信息：

```
print('kcid,',xtfb.kcid,',nday,',nday)
print('preVar,',xtfb.preVars)
print('staVar,',xtfb.staVars)
```

这些输出信息对于建模分析实用价值很大，后面的案例会继续介绍。

运行程序，案例 8-4 的部分输出数据如下：

```
xtfb.poolTrd,足彩推荐
```

	gid	gset	mplay	mtid	gplay	gtid	qj	qs	qr	kend	kwin	kwinrq
tweek	tplay		tsell		cid	pwin9	pdraw9	plost9	kwin_sta			
0	607051	西甲	埃瓦尔	562	格拉纳	4607	-1.0	-1.0	0.0	0	-1.0	
-1.0	1	2017-02-13	2017-02-13	23:55:00	1	1.46	3.75	5.65	3.0			
1	572965	英超	伯恩茅	667	曼城	1072	-1.0	-1.0	0.0	0	-1.0	
-1.0	1	2017-02-13	2017-02-13	23:55:00	1	6.35	4.35	1.35	0.0			
2	581456	澳超	西悉尼	7014	中央海	3616	-1.0	-1.0	0.0	0	-1.0	
-1.0	0	2017-02-12	2017-02-12	13:55:00	1	1.29	4.75	7.00	3.0			
3	610613	意甲	克罗托	243	罗马	1032	-1.0	-1.0	0.0	0	-1.0	
-1.0	0	2017-02-12	2017-02-12	19:25:00	1	7.50	4.60	1.29	0.0			
4	572966	英超	伯恩利	700	切尔西	1173	-1.0	-1.0	0.0	0	-1.0	
-1.0	0	2017-02-12	2017-02-12	21:25:00	1	7.50	4.30	1.31	0.0			


```

xtfb.poolRet, 回报率汇总
      xtim  kret9  kret3  kret1  kret0  knum9  knum3  knum1  knum0
ret9 ...  ret3  ret1  ret0  nwin3  nwin1  nwin0  num3  num1  num0  cid
  1 2017-02-09 101.00  97.75   0.0 114.0  80.00  75.00   0.0
100.00  5.05 ...   3.91   0.0 1.14   3.0   0.0   1.0   4.0   0.0  1.0  1
  2 2017-02-08 106.33  98.20   0.0 147.0  83.33  80.00   0.0
100.00  6.38 ...   4.91   0.0 1.47   4.0   0.0   1.0   5.0   0.0  1.0  1
  3 2017-02-07  78.80  98.50   0.0   0.0  60.00  75.00   0.0   0.00
3.94 ...   3.94   0.0 0.00   3.0   0.0   0.0   4.0   0.0   1.0   1
  4 2017-02-06 130.14 130.14   0.0   0.0 100.00 100.00   0.0
0.00  9.11 ...   9.11   0.0 0.00   7.0   0.0   0.0   7.0   0.0   0.0  1
  5      sum 108.31 111.09   0.0  87.0  84.62  86.96   0.0  66.67
28.16 ...  25.55   0.0 2.61  20.0   0.0   2.0  23.0   0.0   3.0   1

[5 rows x 22 columns]
kcid, 1 ,nday, 5
preVar, []
staVar, [1.5]

#5,backtest,tim:4.40 s

```

输出数据的最后几行是显示部分回溯分析的调用参数。

8.4.4 彩票池与统计池

前面讲过，TFB 极宽足彩量化分析系统借用金融量化的股票池概念，导入了两个全局变量：

- tfsys.gids (-pools)，比赛数据内存数据；
- tfsys.xdats (-pools)，赔率数据内存数据库。

变量名后面括号内的 pools 是为了简化代码，所以进行了省略。

同样，在 tfsys.zTopFootball 类定义中，还定义了两个类似的 pool 变量，用于保存回报数据和下单交易数据。

- poolTrd，下单交易数据。
- poolRet，回报数据记录。

8.4.5 poolTrd 下单交易数据

poolTrd 下单交易数据的数据格式参见 tfsys 模块的 poolSgn:

```
poolSgn=['gid','gset','mplay','mtid','gplay','gtid','qj','qs','qr',
'kend','kwin','kwinrq','tweek','tplay','tsell','cid','pwin9','pdraw9',
'plost9','kwin_sta']
```

各个字段名称的说明如下。

- gid, 比赛场次 id 编码, 是系统唯一的 id。
- gset, game-set 的缩写, 联赛名称。
- mplay, main-play 的缩写, 主队名称。
- mtid, main-team-id 的缩写, 主队的编码 id。
- gplay, guest-play 的缩写, 客队名称。
- gtid, guest-team-id 的缩写, 客队的编码 id。
- qj, 进球的拼音缩写, 这个是国内行业惯例。
- qs, 失球的拼音缩写, 这个是国内行业惯例。
- qr, 让球的拼音缩写, 这个是国内行业惯例。本参数暂未使用, 供日后扩展使用。
- kend, key for end 的缩写, 1 代表比赛完结, 有些取消的比赛其值一直是零。
- kwin, 比赛胜负情况, 3 为主队胜, 1 为平局, 0 为客队胜, 默认和取消的比赛数值是-1。
- kwinrq, 让球的比赛胜负情况, 3 为主队胜, 1 为平局, 0 为客队胜, 默认和取消的比赛数值是-1; 本参数暂未使用, 供日后扩展使用。
- tweek, time for week 的缩写, 比赛周几换算, 一般周末比赛最多。
- tplay, time for play 的缩写, 比赛日期。
- tsell, time for sell 的缩写, 彩票截止销售时间。
- cid, 博彩机构代码, 一般是 1, 代表中国体彩中心。
- pwin9, cid 对应的主队胜收盘赔率。
- pdraw9, cid 对应的平局收盘赔率。
- plostd9, cid 对应的主队负收盘赔率。
- kwin_sta, 系统预测比赛结果, 3 为胜, 1 为平, 0 为负, 默认是-9, 表示无推荐数据。

有些细心的读者可能已经看出来了, poolTrd 下单交易数据的数据格式与 gid 比

赛基本数据的格式类似。的确，基本上 poolTrd 的数据格式就是在 gid 格式的基础上扩充了 cid 博彩机构代码和对应的收盘赔率数据而成的。

需要注意的是最后一条：

kwin_sta, 系统预测比赛结果，3 为胜，1 为平，0 为负，默认是-9 无推荐数据。

kwin_sta 是 TFB 极宽足彩量化分析系统的目标，也是所有人工智能、机器学习的终极目标：预测未来。

所有的人工智能、机器学习算法，目的只有一个，就是通过对历史数据的分析、挖掘，对未来数据进行推荐与预测。

8.4.6 poolRet 回报记录数据

poolRet 回报记录数据的数据格式参见 tfsys 模块的 retSgn:

```
retSgn=['xtim', 'kret9','kret3','kret1','kret0',
'knum9','knum3','knum1','knum0', 'ret9','num9','nwin9',
'ret3','ret1','ret0', 'nwin3','nwin1','nwin0', 'num3','num1','num0']
```

各个字段名称的说明如下。

- xtim, 比赛日期, xtim 的代码是 sum, 表示汇总数据。
- kret9, 总回报率, 这个是按实际投入金额计算的。
- kret3、kret1、kret0, 买入主队胜、平、负盘的回报率。
- knum9, 买对的比赛的总盘数。
- knum3、knum1、knum0, 买入主队胜、平、负盘正确的盘数百分比。
- ret9, 总的回报金额, 包括投入本金。
- num9, 总共下注的比赛盘数。
- nwin9, 总共买对的比赛盘数。
- ret3、ret1、ret0, 买入主队胜、平、负盘分别的总金额。
- nwin3、nwin1、nwin0, 买入主队胜、平、负盘正确的盘数。
- num3、num1、num0, 买入主队胜、平、负盘总的盘数。

以上的回报率都是按单关计算的, 转换成常见的二串一, 通常需要按 90%~95% 的比例进行折算。

8.4.7 实盘足彩推荐分析

下面结合案例 8-4，进行具体的实盘足彩推荐数据分析。

案例 8-4 的足彩推荐数据如下：

xtfb.poolTrd, 足彩推荐

	gid	gset	mplay	mtid	gplay	gtid	qj	qs	qr	kend	kwin	kwinrq
tweek	tplay			tsell			cid	pwin9	pdraw9	plost9	kwin_sta	
0	607051	西甲	埃瓦尔	562	格拉纳	4607	-1.0	-1.0	0.0	0	-1.0	
-1.0	1	2017-02-13	2017-02-13	23:55:00	1	1.46	3.75	5.65	3.0			
1	572965	英超	伯恩茅	667	曼城	1072	-1.0	-1.0	0.0	0	-1.0	
-1.0	1	2017-02-13	2017-02-13	23:55:00	1	6.35	4.35	1.35	0.0			
2	581456	澳超	西悉尼	7014	中央海	3616	-1.0	-1.0	0.0	0	-1.0	
-1.0	0	2017-02-12	2017-02-12	13:55:00	1	1.29	4.75	7.00	3.0			
3	610613	意甲	克罗托	243	罗马	1032	-1.0	-1.0	0.0	0	-1.0	
-1.0	0	2017-02-12	2017-02-12	19:25:00	1	7.50	4.60	1.29	0.0			
4	572966	英超	伯恩利	700	切尔西	1173	-1.0	-1.0	0.0	0	-1.0	
-1.0	0	2017-02-12	2017-02-12	21:25:00	1	7.50	4.30	1.31	0.0			

前面讲过，在推荐信息中，要注意最后一列数据 **kwin_sta**，它是系统预测的比赛结果，3 为胜，1 为平，0 为负，默认是-9，表示无推荐数据。

这个 **kwin_sta** 推荐结果要结合 **kwin** 实际比赛的结果来看，如果两个结果一致，说明预测是正确的。

8.4.8 实盘回报分析

下面结合案例 8-4 进行具体的回报数据分析。

案例 8-4 的回报率汇总数据如下：

xtfb.poolRet, 回报率汇总

	xtim	kret9	kret3	kret1	kret0	knum9	knum3	knum1	knum0	
ret9 ...	ret3	ret1	ret0	nwin3	nwin1	nwin0	num3	num1	num0	cid
1	2017-02-09	101.00	97.75	0.0	114.0	80.00	75.00	0.0		
100.00	5.05 ...	3.91	0.0	1.14	3.0	0.0	1.0	4.0	0.0	1.0 1
2	2017-02-08	106.33	98.20	0.0	147.0	83.33	80.00	0.0		
100.00	6.38 ...	4.91	0.0	1.47	4.0	0.0	1.0	5.0	0.0	1.0 1

3	2017-02-07	78.80	98.50	0.0	0.0	60.00	75.00	0.0	0.00			
3.94	...	3.94	0.0	0.00	3.0	0.0	0.0	4.0	0.0	1.0	1	
4	2017-02-06	130.14	130.14	0.0	0.0	100.00	100.00	0.0				
0.00	9.11	...	9.11	0.0	0.00	7.0	0.0	0.0	7.0	0.0	0.0	1
5	sum	108.31	111.09	0.0	87.0	84.62	86.96	0.0	66.67			
28.16	...	25.55	0.0	2.61	20.0	0.0	2.0	23.0	0.0	3.0	1	

最后一行代码：

5	sum	108.31	111.09	0.0	87.0	84.62	86.96	0.0	66.67			
28.16	...	25.55	0.0	2.61	20.0	0.0	2.0	23.0	0.0	3.0	1	

xtim 数据列的赋值是 sum，表示这一行是汇总数据。

粗略一看，kret 总的回报率是 108.31%，kret3 的买胜局回报率更是高达 111.09%，10%以上的平均回报率，足彩比赛是按日结算的，如果折合成月回报、年回报，那都是天文数字，感觉财务自由就在眼前。

读者不要激动，再看看总的比赛盘数 num9，总共才 26 场比赛，在如今大数据时代，区区的 26 场比赛完全没有说服力。

这个 num9 的数据很正常，因为案例 8-4 只进行 5 天的数据回溯，数据汇总时，需要删除尚未完成的比赛和符合策略函数的比赛，才能纳入统计数据中。

8.4.9 全数据分析与足彩数据集

现在都在讲大数据，而且动不动单位就是以亿计算，本书前面也提过，从 2010 年至今，我国最大的 500 彩票网站收录的比赛数据也不过近 7 万场，主要的赔率数据大约 230 万条。

这么点数据是不是就无法超越大数据理论，进行数据分析呢？

恰恰相反，就笔者个人来看，这么多的足彩数据足够了，特别是对于个人用户来说。

大数据的确越多越好，但大数据多了，就有一个先天的问题，目前的计算机计算能力远远跟不上大数据的成长速度，所以很多大数据项目只能通过数据抽样的方法来解决。

数据抽样，不管如何科学合理，终究是抽样，与全数据存在误差。

而足彩数据规模不大，全盘赔率数据也不过两百多万条，无需抽样，可以直接

进行全数据分析，这是个难得的数据样本。

事实上，就数据规模而言，除了金融股票数据，很少有持续多年的数据范本可以用于数据分析，网站的 log 数据规模虽然偏大，但大部分是人造数据，不是这种原始的经济数据。

所以，从全数据分析这个角度而言，笔者认为足彩数据是最适合数据分析、理论建模的数据集合。

8.5 bt_main回溯主函数

前面已经讲过，真正的回溯分析是由 tfb_backtest 回溯模块的 bt_main 回溯函数完成的，bt_main 函数是整个 TFB 极宽足彩量化分析系统的核心与要点，所以特意用单独一节来进行分析。

bt_main 回溯主函数代码如下：

```
def bt_main(xtfb,timStr):
    if timStr=='':ktim=xtfb.tim_now
    else:ktim=arrow.get(timStr)
    #
    nday=tfsys.xnday_down
    #
    tfsys.gids['kwin_sta']=-1
    #df['kwin_sta']=df['kwin_sta'].astype(int)

    xtfb.poolRet=pd.DataFrame(columns=tfsys.retSgn)
    for tc in range(-3,nday):
        xtim=ktim.shift(days=-tc)
        xtimStr=xtim.format('YYYY-MM-DD')
        print('\n',tc,'#',xtimStr)
        #
        if xtim<xtfb.tim0_gid:break
        #
        xtfb.ktimStr=xtimStr
        bt_1dayMain(xtfb)
```

如图 8-4 所示是 bt_main 回溯主函数的流程图。

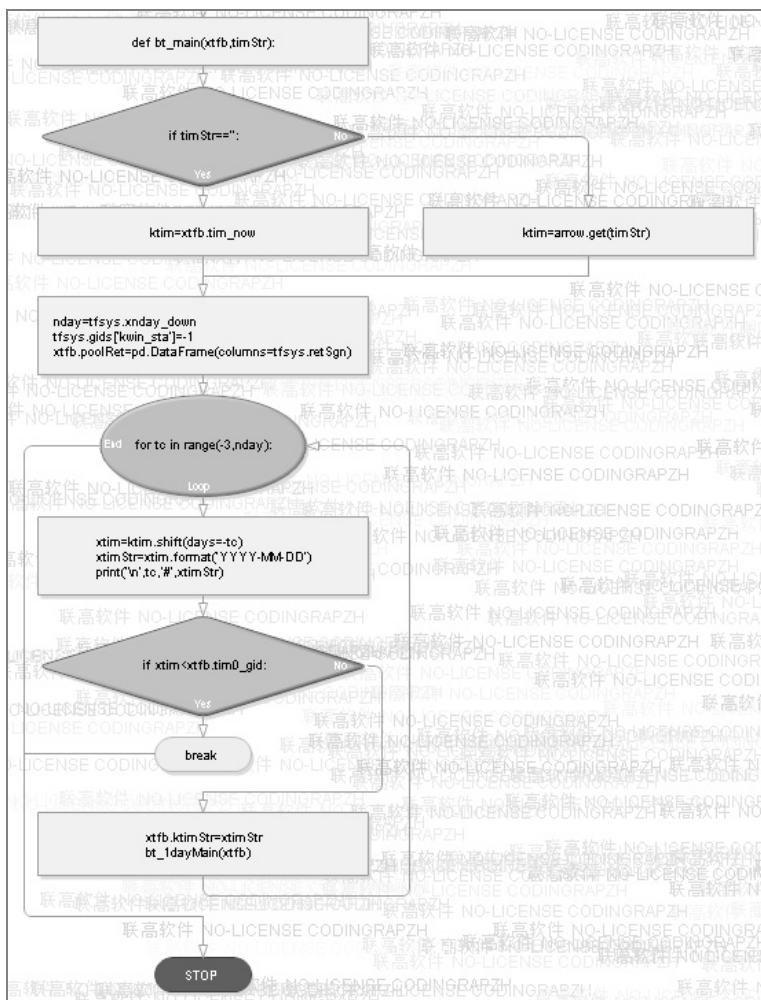


图 8-4 bt_main 回溯主函数流程图

bt_main 回溯主函数的结构很简单，基本流程如下。

- 设置回溯时间参数变量 ktim。
- 设置回溯天数变量 nday。
- 对需要回溯的 gids 内存数据库变量增加一个 kwin_sta 胜负预测数据字段，并初始化为-9。
- 初始化 xtfb.poolRet 回报记录变量，全部清理有关数据。
- 采用 for 循环，调用 bt_1dayMain 单日回溯分析函数。

需要注意的是，`bt_main` 回溯主函数中的 `for` 循环，不是从常规的 0 下标开始，而是从 -3 开始：

```
for tc in range(-3, nday):
```

这是因为足彩实盘采用的是预售模式，-3 表示可以对未来 3 天的比赛数据进行量化分析，这种对未来尚未开始的比赛的分析，就是足彩预测。

而对于已经结束的比赛进行历史数据回溯分析，就属于量化回溯分析，回溯的目的也是为了预测，通过分析历史数据与模型的具体差异，不断优化、调整模型和参数，以获取更好的回报率。

8.5.1 bt_1dayMain 单日回溯函数

通过对 `bt_main` 回溯主函数的分析，我们发现，真正的回溯是通过 `bt_1dayMain` 单日回溯函数完成的。

事实上，TFB 足彩量化分析系统到这里，有两种不同的工程方案，一种是纯粹按 `gid` 比赛编码 `id` 进行回溯测试，另一种是目前采用的按日期模式进行回溯。

传统的金融量化不存在类似的情况，最接近的也不过是彩票池，如上证 50、沪深 300 之类的组合数据。

按 `gid` 比赛编码回溯有一个好处，就是可以简化一级机构，一步到位，因为按日期回溯最终也是需要对当天的比赛按比赛 `id` 进行循环分析。

增加一个单日回溯环节，看起来好像增加了一个环节，但在实盘中，都是按天存在的，无论是投注还是结算，整体而言，反而简化了总体设计，回溯预测一套架构就完全解决了。

此外，采用日结算模式，与传统金融量化分析一样，可以借助很多传统金融程序方面的资源。

`bt_1dayMain` 单日回溯函数代码如下：

```
def bt_1dayMain(xtfb):
    xtfb.poolInx, xtfb.xdat10=[], None
    xtfb.poolDay=pd.DataFrame(columns=tfsys.poolSgn)
    #
    df=tfsys.gids
    g10=df[df.tplay==xtfb.ktimStr]
```



```

#----- lnk.xdat
xdat,xtfb.gid10=bt_lnkXDat(g10,xtfb.kcid)
if len(xdat.index)>0:
    #--dat.pre0
    xlst=['pwin0','pdraw0','plost0','pwin9','pdraw9','plost9']
    tft.fb_df_type2float(xdat,xlst)
    tft.fb_df_type_xed(xdat)
    #
    xtfb.xdat10=xdat
    #
    #2 data.pre
    xtfb.funPre(xtfb)
    #fss='tmp/df901b.dat'
    #xtfb.xdat10.to_csv(fss,index=False,encoding='gb18030')
    #
    #3 bt.anz&trade #gid -->pool
    bt_1d_anz(xtfb)

```

bt_1dayMain 单日回溯函数很简单，运行流程基本如下。

- 初始化 poolInx 比赛编码变量、xdat10 赔率数据和 poolDay 单日交易数据记录等。
- 调用 bt_lnkXDat 函数，按 gid 编码读取单日各场比赛的赔率数据文件，并合并到数据变量 xdat，同时进行简单的清理。
- 如果当天有合适的比赛赔率数据，就先调用 funPre 数据预处理函数，按策略函数的要求对数据进行预处理，如果无需预处理，则调用 sta00_pre 数据预处理函数。
- 调用 bt_1d_anz 单日回报分析函数对数据进行分析，并且保存相应的结果。

8.5.2 赔率数据合并函数

bt_lnkXDat 赔率数据合并函数按 gid 编码读取单日各场比赛的赔率数据文件，并合并到数据变量 xdat，同时进行简单的清理。

bt_lnkXDat 函数代码如下：

```

def bt_lnkXDat(g10,kcid):
    g20=pd.DataFrame(columns=tfsys.gidSgn)

```

```
df9=pd.DataFrame(columns=tfsys.gxdatsgn)
for i, row in g10.iterrows():
    gid=row['gid']
    fxdat=tfsys.rxdatsgn+gid+'_oz.dat'
    if os.path.exists(fxdat):
        df=pd.read_csv(fxdat,index_col=False, dtype=str,
encoding='gb18030')
        c10=df[df.cid==kcid]

        if len(c10.index)==1:
            df9=df9.append(df)
            g20=g20.append(row.T,ignore_index=True)

#
#fss='tmp/df901.dat'
#df9.to_csv(fss,index=False,encoding='gb18030')
#
return df9,g20
```

如果 g10 的数据是 gid2017 所有比赛的数据，则合并后的数据就是 2010—2017 年历年比赛的赔率数据合集。在实盘中，对于赔率数据的合并，也是使用类似方法操作的。

8.5.3 单日回报分析函数

bt_1d_anz 单日回报分析函数名中的 1d 是 1day 的缩写，代码如下：

```
def bt_1d_anz(xtfb):
    # 1#day
    #
    for i, row in xtfb.gid10.iterrows():
        xtfb.bars=row
        #
        bt_1d_anz_1play(xtfb)
    #
    if len(xtfb.poolDay.index)>0:
        ret01=bt_1d_ret(xtfb)
        if ret01['num9']>0:
```

```
xtfb.poolRet=xtfb.poolRet.append(ret01.T,ignore_index=True)
#print(xtfb.poolRet)
```

bt_1d_anz 单日回报分析函数非常简单，分为两大部分。

首先采用迭代模式，对单日所有的比赛按 gid 进行迭代循环，然后调用单场比赛分析函数 bt_1d_anz_1play 进行分析。

需要注意的是，第一部分的代码虽然只有 3 行，但使用了很多专业技巧，如迭代循环，还有以下代码：

```
xtfb.bars=row
```

使用了 bars 数据节点模式，传递单场的比赛数据和赔率数据。

bt_1d_anz 函数后面的部分更简单，就是调用 bt_1d_ret 单日回报计算函数：

```
ret01=bt_1d_ret(xtfb)
```

并把结果追加到 poolRet 回报记录数据池中。

8.5.4 单日回报分析

单日回报分析函数 bt_1d_ret 的代码如下：

```
def bt_1d_ret(xtfb):
    #print('\n@bt_1d_ret')
    ret1d=pd.Series(tfsys.retNil,index=tfsys.retSgn)
    ret1d['xtim'],ret1d['cid'],ret1d['num9']=xtfb.ktimStr,xtfb.kcid,0
    #
    df9=xtfb.poolDay;
    xnum=len(df9.index)
    if xnum==0:return ret1d
    #
    nlst,flst=['kwin','kwin_sta'],['pwin9','pdraw9','plost9']
    tft.fb_df_type4mlst(df9,nlst,flst)
    #
    for i, row in df9.iterrows():
        kwin,kwin2=row['kwin'],row['kwin_sta']
        if kwin>-1:
            rsgn='num'+str(kwin2)
            ret1d['num9'],ret1d[rsgn]=ret1d['num9']+1,ret1d[rsgn]+1
        #
```

```

        if kwin==kwin2:
            dmoney=tft.fb_kwin2pdat(kwin,row)
            rsgn='nwin'+str(kwin2)
            retld['nwin9'],retld[rsgn]=retld['nwin9']+1,retld
[rsgn]+1

            rsgn='ret'+str(kwin2)
            retld['ret9'],retld[rsgn]=retld['ret9']+dmoney,retld
[rsgn]+dmoney

            #print(i,'#1',kwin,dmoney)

#
xlst=[9,3,1,0]
for xd in xlst:
    xss=str(xd)
    dn=retld['num'+xss]
    if dn>0:
        retld['kret'+xss]=round(retld['ret'+xss]/dn*100,2)
        retld['knum'+xss]=round(retld['nwin'+xss]/dn*100,2)

#
#print(retld);
return retld

```

单日回报分析函数 `bt_1d_ret` 的代码虽然看起来很长，但结构很简单，主流程如下。

- 设置单日的回报变量 `retld`，并进行初始化清零。
- 按保存的 `poolDay` 单日推荐比赛数据池，采用迭代的方式提取并计算相关数据。
- 在回溯测试模式下，比赛推荐结果 `kwin_sta` 与实际比赛结果 `kwin` 无论是否相同，都分别按总数和胜、平、负三种模式进行累计。
- 对叠加的数据进行回报率、胜率分析。

需要注意的是：

- TFB 系统中的总回报率，是总的投入金额与总投入盘数的比值，比简单的计算各日的总平均值更加精确；
- 足彩实盘虽然每注是 2 元，但赔率是按 1 元模式设计的，所以在计算回报率时，都是按 1 元模式计算的，与实盘没有差别。

8.5.5 单场比赛回报分析

单场比赛分析回报函数 `bt_1d_anz_1play` 的代码如下：

```
def bt_1d_anz_1play(xtfb):
    #print('\nbt_1d_anz_1play')
    bars=xtfb.bars
    gid=bars['gid']
    xtfb.kgid=gid
    df=xtfb.xdat10[xtfb.xdat10.gid==gid]
    xkwin=xtfb.funSta(xtfb,df)
    #---trade
    if xkwin!=-9:
        xtfb.poolInx.append(gid)
        #
        g10=bars
        c10=df[df.cid==xtfb.kcid]
        #
        g10['kwin_sta'],g10['cid']=xkwin,xtfb.kcid
        g10['pwin9'],g10['pdraw9'],g10['plost9']=c10['pwin9'][0],
        c10['pdraw9'][0],c10['plost9'][0]
        #
        xtfb.poolDay=xtfb.poolDay.append(g10.T,ignore_index=True)
        xtfb.poolTrd=xtfb.poolTrd.append(g10.T,ignore_index=True)
    #print(xtfb.poolDay)
```

运行流程如下。

- 按 `gid` 提取赔率数据到变量 `df`。
- 调用 `funSta` 指定的策略函数，对单场的比赛进行分析，返回预测结果到变量 `xkwin`，如果是-9，则表示系统没有推荐。
- 变量 `xkwin` 值如果是-9，则表示该场比赛系统没有推荐。
- 变量 `xkwin` 值如果不是-9，则表示有推荐，把预测的推荐结果保存到数据列 `kwin_sta`。
- 如果有推荐，还会把相关数据追加到每日比赛汇总数据变量 `poolDay` 和总的下单交易数据变量 `poolTrd` 中。

8.6 sta01策略的大数据分析

案例 8-4 调用的代码是：

```
timStr='2017-02-10'
tfb_main.main_bt(timStr,5)
```

其中，nday=5，前面讲过这个数字太小，无法进行大数据分析，而足彩比赛的总数据不过二百多万条，即使是普通的计算机、笔记本电脑也能够进行全数据分析。

下面结合具体的案例来分析 TFB 策略的运行结果。

案例 8-5 的文件名是 zc805_bt_n3k.py，核心代码如下：

```
fss='dat/poolRet_n3k11.csv'
df=pd.read_csv(fss,index_col=False,encoding='gbk')
df=df[:-1]
df2=df.tail(50)
#
print(df.tail())
print(df.describe())
#
df2[['kret9','kret3','kret0']].plot()
```

案例 8-5 属于数据分析中的二次分析，就是对案例 8-4 或者其他量化分析的 poolRet 回报率数据文件进行再一次分析。

案例 8-5 也是笔者在 TFB 极宽量化分析系统中进行实盘操作时使用的工程程序。

案例 8-5 中的数据文件就是 nday=3000 的全数据回溯测试结果，前面的案例中也讲过，2010—2017 年直接的 nday 不过 2500 多天，设置为 3000 天足够覆盖所有的历史数据日期。

数据文件名 dat/poolRet_n3k11.csv 中的 n3 是 3000d 的意思，k11 表示采用的策略参数是 1.1，这里的策略指的是策略函数 tfsty.sta01_sta，对应 main_bt 函数中的相关代码是：

```
xtfb.funPre=tfsty.sta00_pre
xtfb.funSta=tfsty.sta01_sta
xtfb.preVars=[]
xtfb.staVars=[1.1]
```

案例 8-5 中的以下代码：

```
print(df.tail())
```

可以输出最后的 sum 汇总数据，如图 8-5 所示为数据汇总截图。

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
xtim	kret9	kret3	kret1	kret0	knum9	knum3	knum1	knum0	ret9	num9	nwin9	ret3	retret0	nwin3	nwin1	nwin0	num3	num1	num0	cid	
sum	87.2	89.3	0	77.7	79.8	81.9	0	70.8	112	129	103	93.8	0	18.7	86	0	17	105	0	24	1

图 8-5 数据汇总截图

由图 8-5 所示的数据汇总截图可以看到，进行全数据回溯测试后，tfsty.sta01_sta 策略函数的参数为 1.1 时，汇总结果如下：

- ret9，总的投资回报率是 87.2%，也就是说亏损率在 22% 左右；
- ret3，胜局的回报率是 89.2%，也就是说亏损率在 22% 左右。

看过案例 8-5 的分析结果，读者可以冷静了，虽然策略 sta01_sta 在 5 日的回溯数据分析中有超过 100% 的盈利表现，但是在全数据分析中，总体还是处于亏损状态。

案例 8-5 中的以下代码：

```
df2=df.tail(50)
```

用于截取最后的 50 条回报数据以便绘图，如图 8-6 所示为足彩回报率日线图。

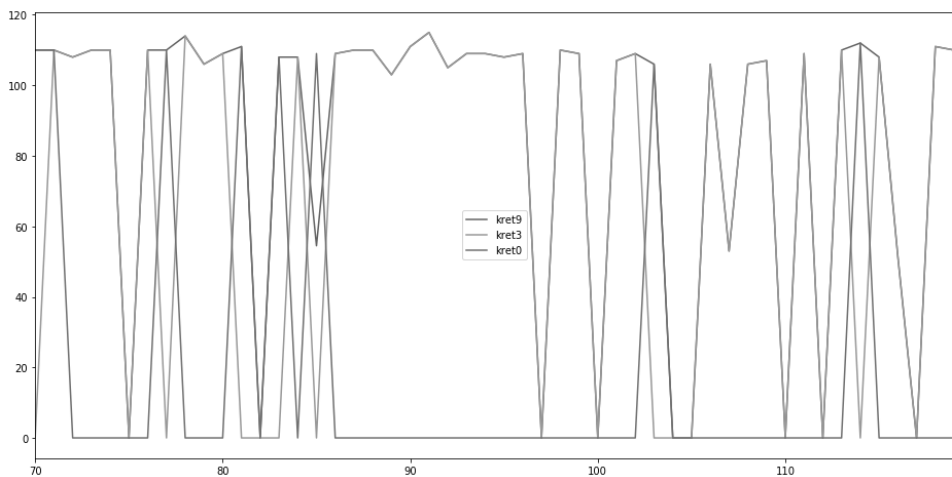


图 8-6 足彩回报率日线图

如果不添加以下代码：

```
df2=df.tail(50)
```

那么在绘图时，部分数据多的图形会混淆。如图 8-7 所示为混淆的足彩回报率日线图，无法看清楚，更不能进行了。

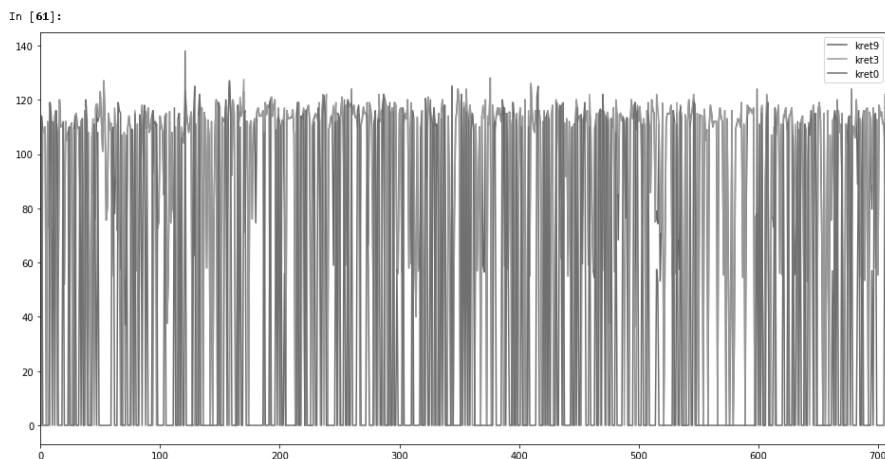


图 8-7 混淆的足彩回报率日线图

8.6.1 一号策略函数

sta01_sta 策略函数也称一号策略函数，一方面是因为 sta01_sta 策略函数简单，很容易作为入门的策略函数模板；另一方面，sta01_sta 策略函数在实盘中，如果操作得当，也是少数能够真正获利、盈利率超过 100% 的实盘策略。

sta01_sta 策略函数位于 tfb_strategy 策略模块库，函数代码如下：

```
def sta01_sta(xtfb, df):
    xkwin, k0 = -9, xtfb.staVars[0]
    #---k0=1.1, k1=80
    df2 = df[df.cid == xtfb.kcid]
    if len(df2.index) > 0:
        dwin, dlose = df2['pwin0'][0], df2['plost0'][0]
        if dwin <= k0: xkwin = 3
        elif dlose <= k0: xkwin = 0
    #
    return xkwin
```

sta01_sta 策略函数实际调用的是回溯模块的 tfbt.bt_1d_anz_1play 单日赔率数据分析函数，调用接口如下：

```
df = xtfb.xdat10[xtfb.xdat10.gid == gid]
xkwin = xtfb.funSta(xtfb, df)
```


xdat10 保存的是回溯当天所有比赛的赔率数据。

sta01_sta 策略函数很简单，首先从 xtfb.staVars[0]策略参数列表中提取参数 k0：

```
xkwin, k0=-9, xtfb.staVars[0]
```

然后根据 kcid 提取对应的赔率数据。

将胜局的赔率和负局的赔率与输入参数 k0 进行比较，代码如下：

```
dwin, dlose=df2['pwin0'][0], df2['plost0'][0]
    if dwin<=k0:xkwin=3
    elif dlose<=k0:xkwin=0
```

sta01_sta 策略函数属于二选一的策略，没有平局的结果，只有流局，即无胜盘推荐，也无负盘推荐。

sta01_sta 策略函数通常输入的 k 值都在 1.5 以下，也就是选择大概率获胜等球队下注，这种策略也称强队策略，虽然胜率大，但是赔率也低，总体的盈利率是属于负值的。

8.6.2 超过 100%的盈利策略与秘诀

前面我们讲过：

sta01_sta 策略函数也称一号策略函数，在实盘中如果操作得当，也是少数能够真正获利、盈利率超过 100%的实盘策略。

在具体分析时，先介绍一种实盘中常见的现象。

目前网络足彩已经禁止，在网络足彩没有被禁止以前，出于营销和吸引用户的考虑，对于 VIP 等大客户，一直有 5%~8%的返点奖励，此外还经常有加奖 10%的活动。

每当有这种活动时，很多大客户都会大笔多批次买入 1.2 以下的低赔率、高胜率的足彩彩票，甚至有人还特意雇人刷票。

那么，这种现象或者行为的背后，到底会不会获利呢？

从表面上看，即使有返点和加奖，按案例 8-5 的分析，进行全数据回溯测试，tfsty.sta01_sta 策略函数的参数为 1.1 时，汇总结果如下：

- ret9，总的投资回报率是 87.2%，也就是说亏损率在 22%左右；
- ret3，胜局的回报率是 89.2%，也就是说亏损率在 22%左右。

如果参数 1.1 不行，那么参数 1.2 如何？我们调整一下参数再进行测试，参数

1.2 的全数据回溯结果文件名是 `dat/poolRet_n3k12.csv`，读者可以自行分析。

如图 8-8 所示是参数为 1.2 的投资回报汇总截图。

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
xtim	kret9	kret3	kret1	kret0	knum9	knum3	knum1	knum0	ret9	num9	nwin9	ret3	ret1	ret0	nwin3	nwin1	nwin0	num3	num1	num0	cid
sum	92.6	92.12	0	94.49	81.29	80.99	0	82.46	1311	1416	1151	1042	0	269	916	0	235	1131	0	285	1

图 8-8 参数为 1.2 的投资回报汇总截图

从回报分析中我们可以看出，策略函数的参数为 1.2 时，回报分析如下：

- `ret9`，总的投资回报率是 92.6%；
- `kret3`、`kret0`，胜盘的投资回报率是 92.12%，负盘的投资回报率是 94.49%；
- `knum9`，总的准确率是 81.29%；
- `knum3`、`knum0`，胜盘的准确率是 80.99%，负盘的准确率是 92.46%。

按以上全数据的回溯分析结果，刷票的总体回报是：

$92.6\% + (5\% \sim 8\%) (\text{返点}) + 81.29\% (\text{胜率}) \times (10\% \sim 15\%) (\text{加奖})$

最终回报率约在 106%~118%之间，扣除本金，约有 6%~18%的盈利空间。

需要注意的是，足彩投资是按日结算的，而且全现金交易，5 万元以下的单注收益无需扣税，即使考虑预售 3~5 天的时间周期，6%~18%的盈利空间即使按最保守的周利润计算，折合成年利率，也是非常恐怖的数十倍了。

足彩具有不确定性，即使在网站举办活动期间，也没有人能够保证一定是正收益，而不会亏本。

基于实盘的经验，这些 VIP 用户通常是足彩行业的“老鸟”、“专家”，所以采用这种多批次、小赔率的操作模式，通过买入尽量多的比赛场次，通过平均概率来降低整体风险。

通过概率投资，而不是通过蓝筹股、绩优股投资，是量化投资与传统投资的一个标志性区别，也可以说是一道分水岭。

读者学量化分析、量化投资，一旦有一天明白了靠概率获利，而不是靠绩优股，不是靠强队、球星、感情来挑选投资对象，就真正步入了量化投资的大门。

资本领域有句格言：大资金有大智慧。

即使在民间，在足彩行业，也不乏资本的智慧之火在闪耀。

8.6.3 统计分析

案例 8-5 中的统计分析非常简单，直接调用了 Pandas 内置的 `describe` 汇总函数：

```
print(df.describe())
```

对应的输出数据如图 8-9 所示。

	kret9	kret3	kret1	kret0	knum9	knum3	knum1	knum0	ret9	num9	...
count	120.000000	120.000000	120.0	120.000000	120.000000	120.000000	120.0	120.000000	120.000000	120.000000	...
mean	88.737500	75.929167	0.0	15.083333	81.250000	69.583333	0.0	13.750000	0.937167	1.075000	...
std	41.419500	49.739460	0.0	37.609355	37.831171	45.511026	0.0	34.276741	0.455438	0.264496	...
min	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	1.000000	...
25%	106.000000	0.000000	0.0	0.000000	100.000000	0.000000	0.0	0.000000	1.060000	1.000000	...
50%	109.000000	108.000000	0.0	0.000000	100.000000	100.000000	0.0	0.000000	1.090000	1.000000	...
75%	110.000000	110.000000	0.0	0.000000	100.000000	100.000000	0.0	0.000000	1.100000	1.000000	...
max	117.000000	115.000000	0.0	117.000000	100.000000	100.000000	0.0	100.000000	2.220000	2.000000	...

	ret3	ret1	ret0	nwin3	nwin1	nwin0	num3	num1	num0	cid
count	120.000000	120.0	120.000000	120.000000	120.0	120.000000	120.000000	120.0	120.000000	120.0
mean	0.781750	0.0	0.155417	0.716667	0.0	0.141667	0.875000	0.0	0.200000	1.0
std	0.514472	0.0	0.384262	0.470711	0.0	0.350170	0.421332	0.0	0.422080	0.0
min	0.000000	0.0	0.000000	0.000000	0.0	0.000000	0.000000	0.0	0.000000	1.0
25%	0.000000	0.0	0.000000	0.000000	0.0	0.000000	1.000000	0.0	0.000000	1.0
50%	1.080000	0.0	0.000000	1.000000	0.0	0.000000	1.000000	0.0	0.000000	1.0
75%	1.100000	0.0	0.000000	1.000000	0.0	0.000000	1.000000	0.0	0.000000	1.0
max	2.220000	0.0	1.170000	2.000000	0.0	1.000000	2.000000	0.0	2.000000	1.0

图 8-9 数据统计截图

因为 poolRet 的数据列较长，无法全部显示，所以图 8-9 中只显示了部分数据，有兴趣的读者可以把 df.describe 的结果保存到文件，再慢慢分析。

8.6.4 回溯时间测试

案例 8-5 中的两个数据文件 dat/poolRet_n3k11.csv 和 dat/poolRet_n3k12.csv，都是使用 nday=3000 的全数据测试模式，只是调用的策略参数不同。

两次回溯测试都对 2010—2017 年近 7 万场比赛数据、近 230 万条赔率数据进行分析，笔者实测时，采用的是 e3-2130 台式机、SSD 硬盘，两次运行时间分别是：

```
#5,backtest,tim:541.24 s
#5,backtest,tim:782.93 s
```

可能后台运行的程序不同，有部分干扰，两次回溯时间差别较大，不过都在 10～15 分钟左右。

这个时间对于量化分析和回溯测试而言已经很快了，金融量化分析通常都是按日来计算的。

这是因为足彩数据规模小，而且采用的策略函数非常简单，如果是本书后面的人工智能、机器学习策略，一场比赛需要数秒甚至几分钟，那么整个回溯周期就很长了。

8.6.5 bt_main_ret 总回报分析

前面已经对回溯分析的各个模块进行了函数级的分析，就差 `bt_main_ret` 总回报分析、数据预处理两个部分。

本节先介绍 `tfb_backtest` 回溯模块中的 `bt_main_ret` 总回报分析函数，函数代码较长，下面分组逐一进行介绍。

函数第 1 组代码如下：

```
def bt_main_ret(xtfb,fgMsg=False):
    #1
    ret9=pd.Series(tfsys.retNil,index=tfsys.retSgn)
    rlst=tfsys.retSgn[1:]
    for rsgn in rlst:
        ret9[rsgn]=xtfb.poolRet[rsgn].sum()
        #print(rsgn,r10[rsgn].sum())
```

定义 `ret9` 回报率变量，并对 `poolRet` 回报数据记录的各条数据进行汇总。

函数第 2 组代码如下：

```
#2
xlst=[9,3,1,0]
for xd in xlst:
    xss=str(xd)
    dn=ret9['num'+xss]
    if dn>0:
        #!!! kret=sum(ret)/num9, not avg (ret)
        ret9['kret'+xss]=round(ret9['ret'+xss]/dn*100,2)
        ret9['knum'+xss]=round(ret9['nwin'+xss]/dn*100,2)
```

对需要计算回报率和胜率的数据列重新进行计算，函数的注解代码如下：

```
#!!! kret=sum(ret)/num9, not avg (ret)
```

再一次强调，这里的回报率是按实际投入总金额计算的，不是各日回报率的简单叠加平均数，更加精确。

函数第 3 组代码如下：

```
#3
nlst=['num9','nwin9','num3','nwin3','num1','nwin1','num0','nwin0']
float_lst=['kret9','kret3','kret1','kret0', 'knum9','knum3','knum1',
'knum0', 'ret9','ret3','ret1','ret0']
```

```
tft.fb_df_type4mlst(xtfb.poolRet,nlst,float_lst)
for xsgn in float_lst:
    xtfb.poolRet[xsgn]=round(xtfb.poolRet[xsgn],2)
    ret9[xsgn]=round(ret9[xsgn],2)
```

函数第3组代码先调用 `tft.fb_df_type4mlst` 对相关的数据格式进行设置，再对相关的回报率数据进行保留两位小数的四舍五入计算，这个环节纯粹为了输出数据美观，不过 `bt_main_ret` 总回报分析函数位于回溯最后阶段，整个回溯已经完成，保存两位数的小数基本够用了。

函数第4组和第5组代码如下：

```
#4
#--save.dat
ret9['xtim'],ret9['cid']='sum',xtfb.kcid
xtfb.poolRet=xtfb.poolRet.append(ret9,ignore_index=True)
xtfb.poolTrd.to_csv(xtfb.poolTrdFN,index=False,encoding='gb18030')
xtfb.poolRet.to_csv(xtfb.poolRetFN,index=False,encoding='gb18030')

#5
if fgMsg:
    print('\nxtfb.poolTrd,足彩推荐')
    print(xtfb.poolTrd.head())
    print('\nxtfb.poolRet,回报率汇总')
    print(xtfb.poolRet.tail())
```

函数第4组和第5组代码很简单，就是保存和显示相关的最终汇总结果。

9

第 9 章

参数智能寻优

参数寻优是人工智能、机器学习的常用术语，就是针对相关的模型寻找最佳的参数组合。

前面的章节提到了 `sta01` 一号策略函数，使用 1.1、1.2 两个不同的参数，最终的回溯测试投资回报率相差很大。

传统的数学模型、量化模型和足彩策略都是依靠经验，通过对实践的不断总结，慢慢积累各种相关的参数。

如今是信息时代、大数据时代，若还依靠人工参数，在效率方面则远远不够，因此就出现了通过计算机程序对相关的参数按目标范围进行逐一测试，从而找到最佳参数的方法。

所谓最佳的参数，对于投资行业和量化分析而言，并非是回报率最大的参数，而是最稳定且又有较高回报率的参数。

例如，对某个参数进行回溯测试，可能有 200%~300% 的回报，但是只有几条数据匹配，没有足够的数据量，很难作为长期投资的依据。另外一个参数，虽然只有 105%（包含本金）的回报率，可是非常稳定，基本上测试数据当中 10% 以上的数据都支持，而且在不同的时间周期测试，回报率也非常稳定、波动很小，这样的模型和参数就是量化分析师梦寐以求的黄金公式。

9.1 一元参数寻优

9.1.1 案例 9-1：一号策略参数寻优

案例 9-1 的文件名是 `zc901_sta01var.py`，基于 `sta01` 一号策略的参数自动寻优程序，只有一个参数，属于最简单的参数寻优案例，核心代码如下：

```
main_var1x()
print('\nok!')
```

主流程很简单，其实就是一条 `main_var1x` 调用代码。

参数寻优程序本身也是工程中采用的代码程序，与 `main_bt` 主入口函数类似，因为参数配置较多，所示采用模板形式，有关函数代码都放在案例主模块文件中。

`main_var1x` 函数是一元参数寻优的主入口函数，代码如下：

```
def main_var1x():
    #1
    mv9=pd.DataFrame(columns=tfsys.btvarSgn)
    flog='tmp/log_v010.csv'
    v1lst=[1.05,1.1,1.15,1.2,1.25,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2]
    nlst=[500,1000,2000,3000]
    #
    for nc in nlst:
        for v1 in v1lst:
            tim0=arrow.now()
            #
            xtfb=main_var100([v1],nday=nc)
            tn=zt.timNSec(' ',tim0)
            r1=xtfb.ret9
            r1['v1'],r1['nday']=v1,v2
            r1['v2'],r1['v3'],r1['v4'],r1['v5']=0,0,0,0
            r1['doc']=str(round(tn))
            #
            mv9=mv9.append(r1.T,ignore_index=True)
            mv9.to_csv(flog,index=False,encoding='gbk')
            print(mv9)
```

`main_var1x` 函数代码分为两组，第 1 组代码定义 `mv9` 测试数据变量，然后设置测试结果保存的 `log` 文件名，相关的测试参数如下：

```
v1lst=[1.05,1.1,1.15,1.2,1.25,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2]
n1st=[500,1000,2000,3000]
```

其中：

- v1lst 是相对 sta01_sta 一号策略函数的调用参数，不同的策略其具体数值不同；
- n1st 用于设置测试周期的参数，这个一元函数基本上都是固定的，采用的是 500 天、1000 天、2000 天、3000 天的模式，因为足彩历史数据目前至少 2500 多天，所以 3000 天是全数据测试，测试时会自动跳过没有数据的日期。

sta01_sta 一号策略函数因为没有数据预处理函数，所以非常简单；如果有数据预处理函数，则需要对预处理函数进行独立的一元参数测试，或者策略函数与预处理函数作为组合，进行二元或者多元参数测试。

main_var1x 函数第 2 组代码是两个 for 循环定义，虽然 Python 语言不推荐 for 循环叠加使用，但这两个 for 循环都是最外层的循环，相对于每个时间周期的计算量，性能影响有限。

我们的一元、二元测试函数都是采用外循环模式，关键是简单，容易看懂程序结构。

测试的主体在循环中，根据循环的各个参数，调用 main_var100 一元测试函数：

```
xtfb=main_var100([v1],nday=nc)
```

然后根据测试结果，记录到 mv9 数据变量中，因为每次测试的周期较长，所以每次测试都保存一次测试结果文件。

案例 9-1 虽然是一元测试，但有两个变量参数，即参数 v1 和时间周期参数 nday。严格地说，也可以称做二元测试。

nday 时间周期参数重点是测试参数的稳定性，各个参数都需要，所以这里没有把其作为独立的一元。

9.1.2 一元测试函数

main_var100 一元测试函数是参数测试的主体，代码如下：

```
def main_var100(v1st,timStr='',nday=2):
    #
    #1---init.sys
    print('\nmain_bt,nday:',nday)
```



```

tfsys.xnday_down=nday
zsys.web_get001txtFg=True

#2---init.tfb
rs0='/tfbDat/'
fgid=rs0+'gid2017.dat'
xtfb=tft.fb_init(rs0,fgid)
if nday== -1:
    tfsys.xnday_down=xtfb.gid_nday+10
    print('nday,',tfsys.xnday_down)

#
#3---backtest
print('\n#3,backtest')
if nday!=0:
    xtfb.funPre=tfsty.sta00_pre
    xtfb.funSta=tfsty.sta01_sta
    xtfb.preVars=[]
    #
    xtfb.staVars=vlst
    #

    xtfb.kcid='1' #cn,3=bet365
    timStr='2017-02-10'
    tfbt.bt_main(xtfb,timStr)

#
#4---main_ret
print('\n#4,result.anz')
tfbt.bt_main_ret(xtfb)
print('kcid,',xtfb.kcid,',nday,',nday)
#print('preVar,',xtfb.preVars)
#print('staVar,',xtfb.staVars)

#
#5
tn=zt.timNSec('',xtfb.tim0,'')
print('\n#5,backtest,tim:{0:.2f} s'.format(tn))
#
#6---end.main

```

```
#
return xtfb
```

下面只介绍改动的部分。

```
def main var100(vlst,timStr='',nday=2):
```

函数第 3 组代码也有改动:

策略函数采用的是 `vlst` 中的数据。

9.1.3 测试结果数据格式

```
        btvarNil=['', 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,'']  
        btvarSgn=['xtim', 'kret9','kret3','kret1','kret0', 'knum9','knum3',  
                  'knum1','knum0', 'ret9','num9','nwin9', 'ret3','ret1','ret0',  
                  'nwin3','nwin1','nwin0', 'num3','num1','num0'  
                  , 'v1','v2','v3','v4','v5','nday','doc']
```

```
r1=xtfb.ret9
r1['v1'],r1['nday']=v1,v2,nc
r1['v2'],r1['v3'],r1['v4'],r1['v5']=0,0,0,0
r1['nday']=str(round(tn))
```

• 236 •

字段，用于保存测试参数。

- v1 至 v5 是一元至五元的参数变量，虽然本案例只使用了一元数据，但在实盘测试中，涉及的变量种类很多，也有一些变化很少，就 2~3 个数值，所以通常采用五元模式保存测试参数，留有部分冗余，也便于日后扩展。
 - nday 是测试时间周期。
 - doc 是注解说明信息，默认是时间周期内测试运行的时间。
- 后面的案例会具体说明以上内容。

9.1.4 案例 9-2：一元参数图表分析

案例 9-2 的文件名是 dat/log_v010.csv，是由案例 9-1 生成的，先使用最常用的数据分析软件 Excel 打开文件查看一下，如图 9-1 所示是选择 kret 总回报率排序后的部分截图。

kma0	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
kma0	kma3	kma9	kret0	kret3	kret9	nday	num0	num3	num9	rw0	rw3	rw9	ret0	ret3	ret9	v1	v2						
0	0	100	100	0	0	105	105	1000	0	0	1	1	0	0	1	1	0	0	1.05	1.05	1.05		
0	0	100	100	0	0	105	105	2000	0	0	1	1	0	0	1	1	0	0	1.05	1.05	1.05		
0	0	100	100	0	0	105	105	3000	0	0	1	1	0	0	1	1	0	0	1.05	1.05	1.05		
86.78	0	83.92	84.5	99.42	0	95.1	95.97	500	121	0	479	600	105	0	402	507	120.3	0	455.53	575.83	1.2		
81.98	0	85.71	84.97	91.92	0	95.43	94.73	1000	111	0	448	559	91	0	384	475	102.03	0	427.51	529.54	1.15		
78.79	0	81.61	80.99	92.6	0	94.27	93.9	500	198	0	707	905	156	0	377	733	183.34	0	696.48	849.82	1.25		
84.91	0	81.59	82.28	97.44	0	92.91	93.84	1000	232	0	891	1123	197	0	727	924	226.06	0	827.81	1053.87	1.2		
84.91	0	84.58	84.64	95	0	93.55	93.81	500	53	0	240	293	45	0	203	248	50.35	0	224.51	274.86	1.15		
78.12	0	78.33	78.28	93.61	0	92.66	92.88	500	289	0	992	1280	225	0	777	1002	293.6	0	919.2	1188.9	1.3		
78.17	0	84.64	83.36	87.51	0	94.11	92.8	2000	142	0	573	715	111	0	485	596	124.26	0	539.24	663.5	1.15		
78.17	0	84.64	83.36	87.51	0	94.11	92.8	3000	142	0	573	715	111	0	485	596	124.26	0	539.24	663.5	1.15		

图 9-1 数据文件截图

由图 9-1 可以看出：

- 虽然前面三条数据的 kret 总回报率高达 105%，但 num9 数据总量只有 1 条，属于偶发事件，没有实盘价值。
- 随后的几条数据，kret 总回报率在 92.8%~95.97% 之间，而且 num9 数据总量除了一条外都在 500 以上，说明有实盘价值。
- 上述几条数据，使用的参数 v1 的取值是 1.15、1.2、1.25、1.3，共四个数值，其中最好的是 1.2。
- 参考 nday 测试周期，可以看到 v1 取 1.15 时 kret9 总回报率最稳定。500 天、1000 天、2000 天、3000 天的测试周期，kret9 总回报率均稳定在前面几位。

综上所述，1.15 是案例 9-1 的最佳参数，也就是说，根据回溯分析，购买 1.15 赔率以下的比赛，配合彩票网站的活动奖励还有返点利润，获得的回报率最稳定，而且获利率也处于相对高位。

案例 9-2 的文件名是 `zc902_sta01vdr.py`，是根据案例 9-1 一号策略的参数自动寻优程序输出的测试数据文件，再进行的二次图表分析，核心代码如下：

```
#1
fss='dat/log_v010.csv'
df=pd.read_csv(fss,index_col=False,encoding='gbk')
#2
#v1lst=[1.05,1.1,1.15,1.2,1.25,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2]
vlst=[1.1,1.15,1.2,1.25,1.3]
nlst=[500,1000,2000,3000]
df['v1']=df['v1'].astype(float)
df=df.set_index(['nday'])
df5=pd.DataFrame()
#3
for v1 in vlst:
    df2=df[df['v1']==v1]
    #print(df2)
    xss='v1_{0:.2f}'.format(v1)
    df5[xss]=df2['kret9']
#4
print(df5)
df5.plot(grid=True)
```

案例 9-2 运行流程如下。

- 第 1 组程序，读入测试数据文件到 `df` 变量。
- 第 2 组程序，根据测试参数列表和 Excel 对结果数据的简单分析，做一个简化的 `vlst` 变量列表，设置其他参数，并定义一个 `df5` 的 Pandas 矩阵 `DataFrame` 变量。
- 第 3 组程序，基于 `vlst` 的循环，提取各组参数的总回报数据，并保存到 `df5`。
- 第 4 组程序，输出 `df5` 的结果，并绘制相关的图形。

案例 9-2 运行结果如下：

	v1_1.10	v1_1.15	v1_1.20	v1_1.25	v1_1.30
nday					
500.0	89.08	93.81	95.97	93.90	92.88
1000.0	91.34	94.73	93.84	92.28	90.72
2000.0	87.18	92.80	92.72	91.30	90.00
3000.0	87.18	92.80	92.72	91.30	90.00

输出的总回报率曲线图如图 9-2 所示。

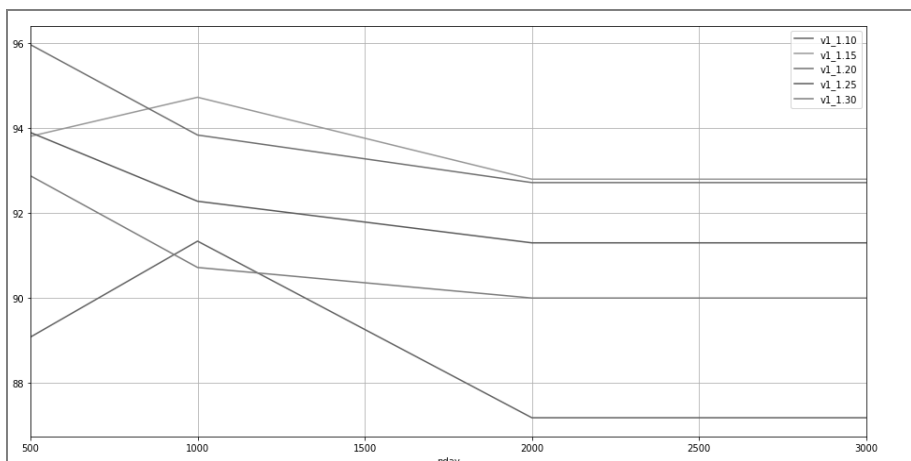


图 9-2 总回报率曲线图

由图 9-2 的图形可以看出,除了参数 1.1 的曲线走势波动较大、回报率低于 88% 外,其他的几个参数总体回报率都在 90% 以上,而且 nday 在 500~3000 的不同时间周期,走势都是趋于稳定的。

总体而言,稳定后的曲线回报率最高的还是参数 1.15,这个结果与我们在 Excel 报表中初步分析的结果相同。

对于出现在循环中的代码:

```
df5[xss]=df2['kret9']
```

稍作修改,就可绘制胜盘回报率的曲线图形:

```
df5[xss]=df2['kret3']
```

以及负盘回报率的曲线图形:

```
df5[xss]=df2['kret0']
```

以下是胜盘数据的输出信息:

	v1_1.10	v1_1.15	v1_1.20	v1_1.25	v1_1.30
nday					
500.0	91.98	93.55	95.10	94.27	92.66
1000.0	94.65	95.43	92.91	92.59	90.70
2000.0	89.34	94.11	92.24	91.78	90.09
3000.0	89.34	94.11	92.24	91.78	90.09

如图 9-3 所示是胜盘回报率的曲线图。

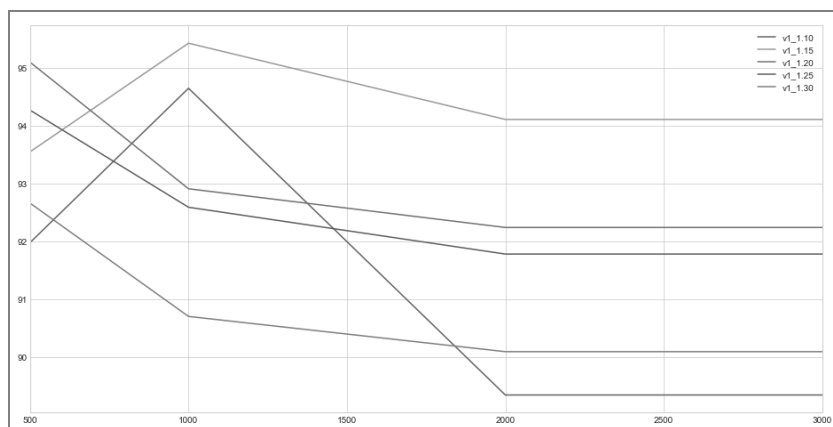


图 9-3 胜盘回报率曲线图

由图 9-3 可以看出，在胜盘曲线中，参数 1.15 的优势更大。

以下是负盘数据的输出信息：

	v1_1.10	v1_1.15	v1_1.20	v1_1.25	v1_1.30
nday					
500.0	72.67	95.00	99.42	92.60	93.61
1000.0	78.43	91.92	97.44	91.14	90.79
2000.0	77.71	87.51	94.62	89.50	89.64
3000.0	77.71	87.51	94.62	89.50	89.64

如图 9-4 所示是负盘回报率的曲线图。

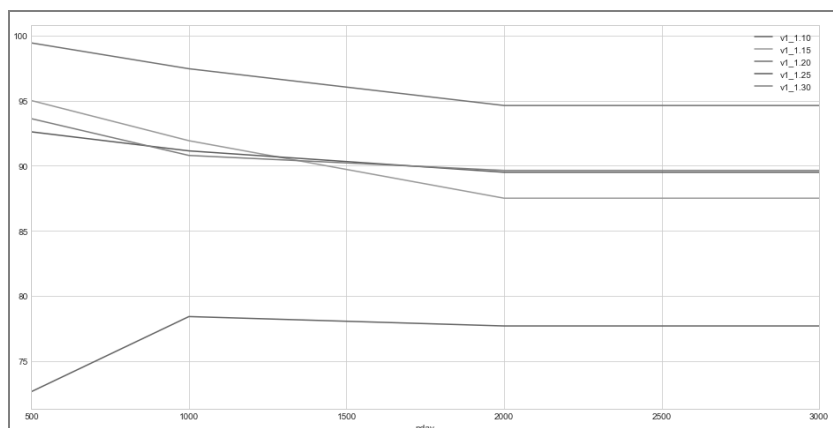


图 9-4 负盘回报率曲线图

由图 9-4 可以看出，在负盘比赛中，走势稳定、回报率最高的是参数 1.2。因此，在实盘中负盘使用 1.2 作为赔率参数，理论上总体回报率更高。

9.2 策略函数扩展

量化分析的核心是策略，而策略源自经验积累和学习扩展，足彩量化分析也是如此。

此外，对现有策略函数进行扩展也是一个方法。

9.2.1 扩展一号策略函数

通过对案例 9-2 进行图表分析发现，胜盘、负盘比赛的最佳回报率参数是不同的，采用不同的赔率基数，理论上总体回报率更高。

关于这个问题，读者可以对 `tfsty.sta01_sta` 策略函数进行扩展，把胜盘和负盘的参数值 `k0` 分为两个参数。

一号策略函数的代码是：

```
def sta01_sta(xtfb,df):
    xkwin,k0=-9,xtfb.staVars[0]
    #---k0=1.1,k1=80
    df2=df[df.cid==xtfb.kcid]
    if len(df2.index)>0:
        dwin,dlose=df2['pwin0'][0],df2['plost0'][0]
        if dwin<=k0:xkwin=3
        elif dlose<=k0:xkwin=0
    #
    return xkwin
```

修改后的策略函数的代码是：

```
def sta01ext_sta(xtfb,df):
    xkwin,k30,k00=-9,xtfb.staVars[0] ,xtfb.staVars[1]
    #---k0=1.1,k1=80
    df2=df[df.cid==xtfb.kcid]
    if len(df2.index)>0:
```

```

        dwin,dlose=df2['pwin0'][0],df2['plost0'][0]
        if dwin<=k30:xkwin=3
        elif dlose<=k00:xkwin=0
    #
    return xkwin

```

9.2.2 案例 9-3：一号扩展策略

案例 9-3 的文件名是 `zc903_sta01x.py`，通过具体的案例，查看扩展后的回报率到底如何，程序核心代码如下：

```

#1
vlst=[1.15,1.2]

#2
funSta=tfsty.sta01_sta
xtfb=main_x100(vlst,funSta,nday=1000)
#
fss='tmp/sta01.csv'
print('nfss',fss)
print(xtfb.poolRet.tail())
xtfb.poolRet.to_csv(fss,index=False,encoding='gbk')

#3
funSta=tfsty.sta01ext_sta
xtfb=main_x100(vlst,funSta,nday=1000)
#
fss='tmp/sta01ext.csv'
print('nfss',fss)
print(xtfb.poolRet.tail())
xtfb.poolRet.to_csv(fss,index=False,encoding='gbk')

```

运行流程如下。

- 第 1 组代码，设置测试参数，虽然有两个参数，但原一号策略函数值只使用一个。
- 第 2 组代码，调用 `main_x100` 函数，使用 `sta01` 策略模式，保存测试结果。
- 第 3 组代码，调用 `main_x100` 函数，使用 `sta01ext` 扩展策略模式，保存测试结果。

为了方便对比不同的策略函数，笔者对测试主入口函数进行了部分修改，函数定义改为以下代码：

```
def main_x100(vlst,staFun,timStr='',nday=2):
```

增加一个 **staFun** 策略函数参数变量，对应的函数内部代码位于函数第 3 组回溯调用部分，代码如下：

```
xtfb.funSta=staFun
```

案例 9-3 的运行结果较长，为了便于阅读，删除了部分无关的数据列，只保存最后的汇总数据，其中 **sta01** 策略的回报率数据如下：

```
kret9 kret3 kret0 knum9 knum3 knum0 ret9 num9 nwin9 ret3 ret0 nwin3 nwin0
num3 num0
94.73 95.43 91.92 84.97 85.71 81.98 529.54 559 475 427.51 102.03 384 91
448 111
```

sta01ext 扩展策略的回报率数据如下：

```
kret9 kret3 kret0 knum9 knum3 knum0 ret9 num9 nwin9 ret3 ret0 nwin3 nwin0
num3 num0
96.11 95.43 97.44 85.44 85.71 84.91 653.57 680 581 427.51 226.06 384 197
448 232
```

对两者进行对比可以发现：

- 扩展策略的回报率是 96.11%，而原策略的回报率是 94.73%，增加了 1.5%，这个提升比例虽然看起来小，却全部是纯利润，相当于把利润 v 提高了 10%~15%，非常可观；
- 扩展策略总胜率是 85.44%，比原策略的 84.97% 高 0.5% 左右；
- 扩展策略在胜盘回报率的胜率方面没有任何变化，说明是纯增益策略；
- 扩展策略负盘回报率是 97.44%，比原策略的 91.92% 高了近 6%；
- 扩展策略负盘胜率为 84.91%，比原策略负盘胜率的 81.98% 高 3% 左右；
- 扩展策略的比赛场次 $\text{num0}=232$ ，比原策略的 111 场多了近 1 倍。

由分析结果可见，**sta01ext** 扩展策略基本上在每一个环节都是正收益，特别是在比赛场次 num0 方面，扩大了数据基数。

通常的优化策略都是采用降级模式，会缩减原策略的数据规模，由此可见，**sta01ext** 扩展策略是一次非常成功的扩展。

9.2.3 案例 9-4: sta10 策略

在介绍多元测试程序之前，先介绍一下 sta10 策略和相关的案例，因为 sta10 函数中使用了两组参数变量。

sta10 策略函数名是 sta10_sta，位于 tfb_strategy 模块，相关代码如下：

```
def sta10_sta(xtfb,df):
    xkwin,k0,k1=-9,xtfb.staVars[0],xtfb.staVars[1]
    #---k0=1.1,k1=80
    df3=df[df.pwin0<k0]
    df0=df[df.plost0<k0]
    xn9=len(df.index)
    if xn9>0:
        kn3,kn0=len(df3.index)/xn9*100,len(df0.index)/xn9*100
        if kn3>k1:xkwin=3
        elif kn0>k1:xkwin=0
    #
    return xkwin
```

函数代码还是很简单的，基本流程如下。

- 初始化 xkwin 变量，提取策略函数所需的参数 k0 和 k1。
- 按 k0 参数提取胜盘机构的数据到 df3 变量，再提取负盘机构的数据到 df0 变量。
- 计算胜盘、负盘机构在机构总数当中所占比例，并保存到变量 kn3 和 kn0 中。
- 把 kn3、kn0 和参数 k1 进行比较，判断是胜盘还是负盘。

sta10 策略函数属于二选一的策略，没有平局判定。

此外，sta10 策略函数为了简化代码，还利用了足彩领域的很多潜规则。

- 参数 k0 默认是 1.5 以下的赔率，属于主队强势比赛。
- 足彩实盘很少有胜盘、负盘都大于 2 或者小于 2 的赔率。
- 足彩行规，60%以上概率属于大概率获胜。

如果没有按照以上实盘规则，只是按照一般程序编程，则 sta10 策略函数还需要补充不少冗余代码，用于判别 kn3、kn0 之间的差值，以及胜盘、负盘赔率的差值等。

案例 9-4 的文件名是 zc904_sta10.py，核心代码如下：

```
timStr='2017-02-10'
main_bt(timStr,50)
print('\nok!')
```

程序很简单，为了保持一致的学习体验，我们设置了 `timStr='2017-02-10'`，以统一测试周期。

此外，为了体现回报率数据，采用的是 `nday=50` 天的测试周期，没有采用默认的 `nday=2` 天。

在 `main_bt` 回溯主函数中，与以往的案例大体相同，不同之处主要在第3组代码，回溯参数设置如下：

```
xtfb.funPre=tfsty.sta00_pre
xtfb.funSta=tfsty.sta10_sta
xtfb.preVars=[]
xtfb.staVars=[1.25,80]
```

数据预处理是空函数，将策略函数设置为 `sta10_sta` 是惯例，最大的不同是以下代码：

```
xtfb.staVars=[1.25,80]
```

`staVars` 中提供两组数值，其中的 1.25 与 80 都是经验参数，到底结果如何，还需要后面的参数寻优进行计算与分析。

案例 9-4 部分输出信息如下：

```
xtfb.poolRet, 回报率汇总
```

	xtim	kret9	kret3	kret1	kret0	knum9	knum3	knum1	knum0	
ret9 ...	ret3	ret1	ret0	nwin3	nwin1	nwin0	num3	num1	num0	cid
14	2016-12-31	112.0	0.00	0.0	112.0	100.00	0.00	0.0	100.0	
1.12 ...	0.00	0.0	1.12	0.0	0.0	1.0	0.0	0.0	1.0	1
15	2016-12-30	0.0	0.00	0.0	0.0	0.00	0.00	0.0	0.0	
0.00 ...	0.00	0.0	0.00	0.0	0.0	0.0	1.0	0.0	0.0	1
16	2016-12-27	105.0	105.00	0.0	0.0	100.00	100.00	0.0	0.0	
1.05 ...	1.05	0.0	0.00	1.0	0.0	0.0	1.0	0.0	0.0	1
17	2016-12-23	109.0	109.00	0.0	0.0	100.00	100.00	0.0	0.0	
1.09 ...	1.09	0.0	0.00	1.0	0.0	0.0	1.0	0.0	0.0	1
18	sum	95.3	90.28	0.0	113.4	86.96	83.33	0.0	100.0	
21.92 ...	16.25	0.0	5.67	15.0	0.0	5.0	18.0	0.0	5.0	1

```
[5 rows x 22 columns]
kcid, 1 ,nday, 50
preVar, []
staVar, [1.25, 80]
```

```
#5,backtest,tim:16.94 s
```

sta10 策略函数使用 50 天的回溯测试，耗时 16.9 秒，还是很快的，其他方面：总的回报率 95%，胜盘 90%，负盘 113%，高于 100% 属于正收益，但是 num0=5，才区区 5 场的比赛数据，数据量有些少，无法作为实盘操作参考。

9.3 二元参数寻优

案例 9-3 因为只进行了 50 天回溯，虽然负盘的回报率高达 113%，可是依然无法作为实盘操作参考。

通常，实盘操作至少需要 100 场以上的数据支持，才有足够的说服力，按案例 9-3 的测试，回溯应该采用 2000 天以上的数据，不过 sta10 是二元策略参数，计算量很大，所以 nday 暂时取 500 天作为测试周期。

9.3.1 案例 9-5: sta10 参数寻优

案例 9-5 的文件名是 zc905_sta10var.py，说明如何利用函数对 sta10 策略进行测试，从而找出最佳的参数组合，核心代码如下：

```
def main_var2x():
    #1
    mv9=pd.DataFrame(columns=tfsys.btvarSgn)
    flog='tmp/log_v020.csv'
    v1lst=[1.05,1.1,1.15,1.2,1.25,1.3,1.4,1.5,1.6,1.7,1.8,1.9,2]
    v2lst=[80,80,60,55,45,40,35]
    #nlst=[500,1000,2000,3000]
    nlst=[500]
    #
    for nc in nlst:
        for v1 in v1lst:
            for v2 in v2lst:
                tim0=arrow.now()
                #
                xtfb=main_var100([v1,v2],nday=nc)
                tn=zt.timNSec('',tim0)
```

```

r1=xtfb.poolRet.tail(1)
r1['v1'],r1['v2'],r1['nday']=v1,v2,nc
r1['v3'],r1['v4'],r1['v5']=0,0,0
r1['doc']=str(round(tn))
#
mv9=mv9.append(r1,ignore_index=True)
mv9.to_csv(flog,index=False,encoding='gbk')
print(mv9)

```

`main_var1x` 函数的代码分为两组，第 1 组定义 `mv9` 测试数据变量，然后设置测试结果保存的 `log` 文件名，相关的测试参数如下：

```

v1lst=[1.05,1.1,1.15,1.2,1.25,1.3,1.4,1.5,1.6]
v2lst=[80,70,60,55,45,40,35]
#n1st=[500,1000,2000,3000]
n1st=[500]

```

其中：

- `v1lst` 和 `v2lst` 是相对 `sta01_sta` 一号策略函数的调用参数，不同的策略具体数值不同；
- `n1st` 用于设置测试周期的参数，因为是二元测试，所以一般不采用多周期模式，而采用 500 天作为测试周期。

`main_var2x` 函数的第 2 组代码是三层 `for` 循环定义，系统中的一元、二元测试函数都采用外循环模式，优点是简单、容易看懂程序结构。

测试的主体在循环中，根据循环的各个参数调用 `main_var100` 回溯测试函数：

```
xtfb=main_var100([v1,v2],nday=nc)
```

注意这里传递的是两个参数 `v1` 和 `v2`，因为采用的是列表变量模式，所以 `main_var100` 回溯测试函数定义部分无需修改。

案例 9-5 测试一组数据大约需要 240 秒，约 4 分钟，`v1` 和 `v2` 两组数据分别有 9 个和 7 个变量：

```

v1lst=[1.05,1.1,1.15,1.2,1.25,1.3,1.4,1.5,1.6]
v2lst=[80,70,60,55,45,40,35]

```

全部测试完大约需要 250 分钟，即 4 个小时左右。

所以二元以上的参数模型，通常都直接使用人工智能、机器学习的模式，不然计算量很大，根本无法用于实盘。

人工智能、机器学习模块库因为内部采用了矢量化运算，而且可以借助 GPU 加

速，所以对于实盘分析来说是必不可少的组成部分。

9.3.2 案例 9-6：二元参数图表分析

案例 9-6 使用的数据文件名是 dat/log_v020.csv，是由案例 9-5 生成的，先使用最常用的 Excel 查看一下，如图 9-5 所示是选择 kret 总回报率排序后的部分截图。

kret9	nday	num0	num1	num3	num9	nwin0	nwin1	nwin3	nwin9	ret0	ret1	ret3	ret9	v1	v2	v
103	500	0	0	1	1	0	0	1	1	0	0	1.03	1.03	1.1	45	
103	500	0	0	1	1	0	0	1	1	0	0	1.03	1.03	1.1	40	
103	500	0	0	1	1	0	0	1	1	0	0	1.03	1.03	1.1	35	
96.87	500	36	0	162	198	32	0	141	173	36.04	0	155.77	191.81	1.2	35	
96.53	500	26	0	132	158	23	0	115	138	25.8	0	126.71	152.51	1.2	45	
96.46	500	27	0	145	172	23	0	127	150	25.8	0	140.11	165.91	1.2	40	
96.19	500	16	0	103	119	14	0	90	104	15.51	0	98.96	114.47	1.2	60	
96.19	500	17	0	109	126	15	0	95	110	16.61	0	104.59	121.2	1.2	55	

图 9-5 数据文件截图

由图 9-5 可以看出：

- 虽然前面三条数据的 kret 总回报率高达 103%，但 num9 数据总量只有 1 条，属于偶发事件，没有实盘价值；
- 随后的几条数据，kret 总回报率非常稳定，都在 96%左右，而且 num9 数据总量每条都在 100 以上，说明有实盘价值，其对应的 v1 参数是 1.2，v2 参数从 35~65 都有；
- 在 v2 的多组参数中，取 35 的 num9 比赛场次最多，基于大数据的大盘原则，v2 参数首选 35。

综上所述：

v1=1.2，v2=35

是本案例相对较佳的参数组合。

案例 9-6 是对以上数据进行二次图表分析，核心源码如下：

```
#1
fss='dat/log_v020.csv'
df=pd.read_csv(fss,index_col=False,encoding='gbk')

#2
#v1lst=[1.05,1.1,1.15,1.2,1.25,1.3,1.4,1.5,1.6]
#v2lst=[80,70,60,55,45,40,35]
#
```

```

v1lst=[1.15,1.2,1.25,1.3,1.4,1.5,1.6]
v2lst=[80,70,60,55,45,40,35]
df['v1']=df['v1'].astype(float)
df['v2']=df['v2'].astype(int)
df=df.set_index(['v2'])
df5=pd.DataFrame()

#3
for v1 in v1lst:
    df2=df[df['v1']==v1]
    #print(df2)
    xss='v1_{0:.2f}'.format(v1)
    df5[xss]=df2['kret9']

#4
print(df5)
df5.plot(grid=True)

```

案例 9-6 运行流程如下。

- 第 1 组程序，读入测试数据文件到 df 变量。
- 第 2 组程序，根据测试参数列表和 Excel 对结果数据的简单分析，做一个简化的 v1lst、v2lst 变量列表，用于设置其他参数，并定义一个 df5 的 Pandas 矩阵 DataFrame 变量。
- 第 3 组程序，基于 v1lst 的循环，提取各组参数的总回报数据并保存到 df5。
- 第 4 组程序，输出 df5 的结果并绘制相关的图形。

案例 9-6 运行结果如下：

	v1_1.15	v1_1.20	v1_1.25	v1_1.30	v1_1.40	v1_1.50	v1_1.60
v2							
80	93.00	90.63	95.19	93.52	92.72	92.29	90.54
70	90.75	93.87	94.92	93.89	93.54	92.06	90.31
60	87.35	96.19	94.75	94.75	93.16	91.77	90.27
55	89.18	96.19	95.41	94.72	93.17	91.62	90.37
45	89.71	96.53	95.46	92.49	93.35	90.97	89.63
40	87.57	96.46	94.42	91.97	93.18	90.95	89.14
35	92.16	96.87	93.99	92.33	93.01	90.95	89.05

输出的总回报率曲线图如图 9-6 所示。

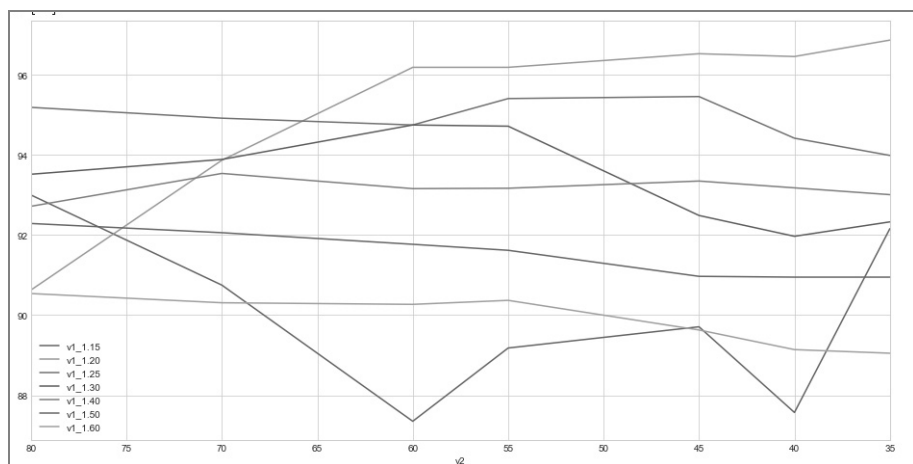


图 9-6 总回报率曲线图

由图 9-6 的图形可以看出，参数 1.15 和 1.6 的曲线走势波动较大，回报率有时低于 90%，而其他的几个参数总体回报率都在 90% 以上。

总体而言，稳定后的曲线回报率最高的还是参数 1.2，v2 在 35~60 的区间中回报率都非常稳定，这个与我们在 Excel 报表中初步分析的结果相同。

对于出现在循环当中的代码：

```
df5[xss]=df2['kret9']
```

稍作修改，就可绘制胜盘回报率的曲线图形：

```
df5[xss]=df2['kret3']
```

以及负盘回报率的曲线图形：

```
df5[xss]=df2['kret0']
```

以下是胜盘数据的输出信息：

	v1_1.15	v1_1.20	v1_1.25	v1_1.30	v1_1.40	v1_1.50	v1_1.60
v2							
80	90.00	90.51	95.06	93.14	92.25	92.14	90.43
70	86.80	93.41	94.64	93.34	92.99	91.71	90.09
60	84.78	96.08	94.02	94.12	92.51	91.73	89.91
55	86.16	95.95	94.70	94.05	92.32	91.51	90.02
45	87.48	95.99	94.68	91.87	92.60	91.01	89.29
40	85.26	96.63	94.03	91.29	92.42	91.25	88.77
35	90.15	96.15	93.53	91.92	92.34	91.04	88.89

如图 9-7 所示是胜盘回报率的曲线图。

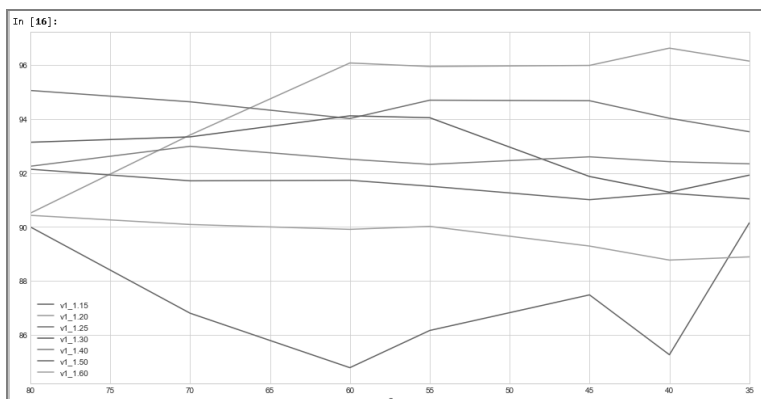


图 9-7 胜盘回报率曲线图

由图 9-7 可以看出，在胜盘回报率曲线中，参数 v_1 取 1.2 的优势更大。

以下是负盘数据的输出信息：

	$v_1_{1.15}$	$v_1_{1.20}$	$v_1_{1.25}$	$v_1_{1.30}$	$v_1_{1.40}$	$v_1_{1.50}$	$v_1_{1.60}$
v_2							
80	111.00	91.83	95.80	95.41	94.46	92.87	90.95
70	110.50	97.67	96.29	96.46	95.57	93.31	91.14
60	110.50	96.94	98.26	97.46	95.52	91.92	91.58
55	108.33	97.71	98.98	97.67	96.19	92.05	91.67
45	108.33	99.23	99.18	95.23	95.99	90.82	90.85
40	108.33	95.56	96.34	94.95	95.86	89.90	90.45
35	109.25	100.11	96.09	93.99	95.39	90.63	89.66

如图 9-8 所示是负盘回报率的曲线图。

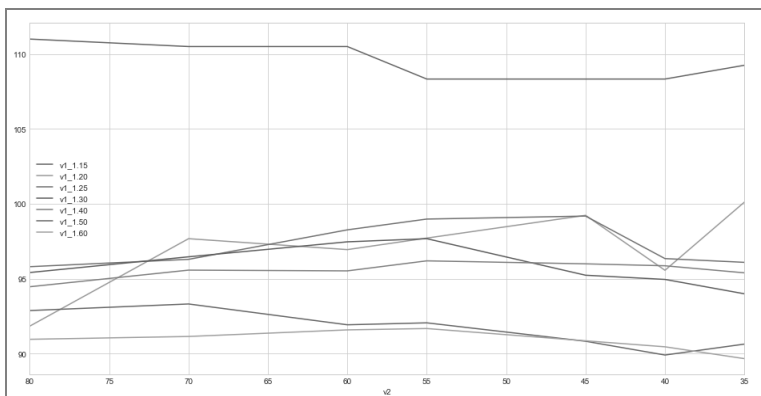


图 9-8 负盘回报率曲线图

由图 9-8 可以看出，在负盘比赛中，除了 v1 取 1.15 的回报率曲线走势有些高得不正常，其他曲线大体走势稳定，回报率最高的是参数 1.2 和参数 1.25。

如图 9-9 所示，再打开数据文件 dat/log_v020.csv，对参数 v1 进行科学排序。

P	Q	R	S	T	U	V	W	X	
nwin0	nwin1	nwin3	nwin9	ret0	ret1	ret3	ret9	v1	v2
1	0	5	6	1.11	0	5.4	6.51	1.15	
2	0	8	10	2.21	0	8.68	10.89	1.15	
2	0	14	16	2.21	0	15.26	17.47	1.15	
3	0	15	18	3.25	0	16.37	19.62	1.15	
3	0	20	23	3.25	0	21.87	25.12	1.15	
3	0	21	24	3.25	0	23.02	26.27	1.15	
4	0	28	32	4.37	0	30.65	35.02	1.15	

图 9-9 数据文件截图

由图 9-9 可以看到，nwin0 负盘获胜的比赛场次还不到 5 个，数据量太少，测试设置的时间周期 nday=500 天，这个数字按日期算，还不到 1%。总体上，最终实盘参考的数据基数要占测试总数的 5%~10%，以保证策略模型的稳定性。

9.4 策略310准多因子策略

策略 310 是对 tfb_strategy 策略模块库中的 sta310_sta 策略函数的简称。策略 310 使用了多组分析策略函数来分析赔率数据，再进行集中汇总分析，计算最终结果。

策略 310 这种复合策略函数，类似传统金融量化的多因子分析，所以这里也称为准多因子分析策略。

sta310_sta 策略函数代码如下：

```
def sta310_sta(xtfb,df):
    xkwin=-9
    xk3,xk1,xk0=sta310_sta3(xtfb,df),sta310_sta1(xtfb,df),sta310_sta0(xtfb,df)
    if (xk3==3)and(xk1<0)and(xk0<1):xkwin=3
    if (xk3<1)and(xk1==1)and(xk0<0):xkwin=1
    if (xk3<1)and(xk1<0)and(xk0==0):xkwin=0
    #
    #print('sta',xkwin,xk3,xk1,xk0)
    return xkwin
```

函数代码很简单，先分别调用 sta310_sta3、sta310_sta1、sta310_sta0 三个策略

子函数，分析比赛的胜、平、负，也就是结果是不是 3、1、0，或者-9 流局，即没有推荐。

其中：

- sta310_sta3 只判断比赛结果是不是胜盘，返回值是 3，或者流局，返回值是-9；
- sta310_sta1 只判断比赛结果是不是平局，返回值是 1，或者流局，返回值是-9；
- sta310_sta0 只判断比赛结果是不是负盘，返回值是 1，或者流局，返回值是-9。

因为每场比赛都只有一个结果，所以如果出现流局以外的两个结果，说明这两个函数的结果互相矛盾，为稳妥起见，sta310_sta 策略函数把矛盾的结果改为流局，返回值是-9。

那么 sta310_sta3、sta310_sta1、sta310_sta0 三个策略子函数又是如何判断比赛结果的呢？

以上三个函数的结构、代码都非常类似，所以只介绍其中的 sta310_sta3 策略子函数，其他两个函数请读者自己解读。

sta310_sta3 策略子函数代码如下：

```
def sta310_sta3(xtfb,df):
    df9=df[df.kpwin>100]
    df1=df[df.kpwin<=100]
    xn= len(df1.index)-len(df9.index)
    #
    xkwin,k0=-9,xtfb.staVars[0]
    if xn>k0:xkwin=3
    #
    return xkwin
```

函数代码还是很简单的，其中的 kpwin、kpdraw、kplost 源自策略 310 的数据与处理函数 sta310_pre，其代码如下：

```
def sta310_pre(xtfb):
    df=xtfb.xdat10
    #
    df['kpwin']=round(df['pwin9']/df['pwin0']*100)
    df['kplost']=round(df['plost9']/df['plost0']*100)
    df['kpdraw']=round(df['pdraw9']/df['pdraw0']*100)
    #
    return df
```

其中，kpwin、kpdraw、kplost 分别是胜局、平局、负局的收盘赔率和开盘赔率

的百分比数值。

想弄清楚 `kpwin`、`kpdrow`、`kplost` 三个变量的来龙去脉，需要解读 `sta310_sta3` 策略子函数。

其中的代码如下：

```
xn=len(df1.index)-len(df9.index)
```

表示在众多博彩机构中，变量 `df9` 胜局收盘赔率上涨（>100%）和变量 `df1` 胜局收盘赔率下跌（>100%）之间的机构数目差。

在实盘中，如果博彩机构在开盘后认为主队获胜的概率增加，则会下调赔率数值，反之会加大赔率数值。

因此，下调赔率数值的机构越多，说明越多机构看好主队胜盘，类似于机构对于个股的分析，只是角度不同。

具体到 `sta310_sta3` 策略子函数，最终评定的代码是：

```
if xn>k0:xkwin=3
```

如果下跌的结构数据和上涨的结构数据之间的差值大于预设的数值 `k0`，则函数认为是上涨。

在采集的数据中，通常有 30 家机构，还要加上平均值、最大值、最小值三组衍生数据，通常 `k0` 的值在 3~5 之间。

当然，策略 310 只是教学策略函数，在实盘中，为了精确，可以把结果数据不足 30 家的比赛过滤掉，以提高数据的准确度。

9.4.1 案例 9-7：数据预处理

`dataPre` 数据预处理模块是极宽量化软件的一大创新，最早源自 `zwQuant` 量化软件。通常，金融产品的数据源只提供 OHLC 和成交量等基本数据，在任何量化回溯分析中，都需要根据时间序列对这些数据进行衍生运算。传统的量化软件，大部分是在策略分析时，再对每个数据进行衍生扩展。

在 C、Java、BASIC 等传统编程语言中，这种模式类似 `lazy`（懒惰运算）模式，对于整体项目的运算速度影响很小。有时，因为 `lazy` 是“按需计算”，甚至还可以提高整体项目的速度。不过对于 Python、Pandas、Numpy、MATLAB 和 GPU 超算来说，这些基于矩阵的现代矢量数据处理框架显然不适合。

Pandas、Numpy 都是经过高度优化的矢量矩阵模块库，以最简单的求和计算为

例，Pandas 只需一个 SUM 函数即可替代 Python 数十行代码，而且速度快上百倍。

至今为止，在数据处理环节，大多使用以下代码：

```
xtfb.funPre=tfsty.sta00_pre
```

使用的都是 sta00_pre，尚未涉及具体数据预处理环节，但我们可以通过其他的策略函数提前介绍一下数据预处理函数。

sta310_pre 函数是 sta310 策略的数据预处理函数，代码如下：

```
def sta310_pre(xtfb):
    df=xtfb.xdat10
    #
    #
    df['kpwin']=round(df['pwin9']/df['pwin0']*100)
    df['kplost']=round(df['plost9']/df['plost0']*100)
    df['kpdraw']=round(df['pdraw9']/df['pdraw0']*100)
    #
    return df
```

下面通过具体的案例来讲解 sta310_pre 数据预处理函数。

案例 9-7 的文件名是 zc9-7_datpre.py，核心代码如下：

```
fss='dat/240228_oz.dat'
xtfb=tft.fb_init()
xtfb.xdat10=pd.read_csv(fss,index_col=False,encoding='gbk')
#
df=tfsty.sta310_pre(xtfb)
#
print('\ndf')
print(df.tail())
print('\nxtfb.xdat10')
print(xtfb.xdat10.tail())
```

为了强调 sta310_pre 数据预处理函数，案例 9-7 省略了冗长的回溯架构，直接调用 sta310_pre 数据预处理函数。

案例 9-7 的输出信息如下：

```
df
      gid   cid   cname  pwin0  pdraw0  plost0  pwin9  pdraw9  plost9
vwin0  ...   mplay  mtid  qj  qr  qs      tplay  tweek  kpwin  kplost
kpdraw
28  240228   413   Bet3000  1.20   6.00   13.00   1.20   6.00   13.00
```

```

77.38 ... 叙利亚 94 4 0 0 2010-03-03 3 100.0 100.0
100.0
29 240228 525 Bet7days 1.23 5.50 10.00 1.23 5.50 10.00
74.26 ... 叙利亚 94 4 0 0 2010-03-03 3 100.0 100.0
100.0
30 240228 90005 gavg 1.19 5.42 11.14 1.18 5.53 12.05
75.07 ... 叙利亚 94 4 0 0 2010-03-03 3 99.0 108.0
102.0
31 240228 90009 gmax 1.25 7.20 18.00 1.23 7.00 18.50
80.60 ... 叙利亚 94 4 0 0 2010-03-03 3 98.0 103.0
97.0
32 240228 90001 gmin 1.11 4.10 5.90 1.11 4.10 8.74
69.46 ... 叙利亚 94 4 0 0 2010-03-03 3 100.0 148.0
100.0

[5 rows x 38 columns]

xtfb.xdat10
      gid  cid  cname pwin0 pdraw0 plost0 pwin9 pdraw9 plost9
vwin0 ... mplay mtid qj qr qs      tplay tweek kpwin kplost
kpdraw
28 240228 413 Bet3000 1.20 6.00 13.00 1.20 6.00 13.00
77.38 ... 叙利亚 94 4 0 0 2010-03-03 3 100.0 100.0
100.0
29 240228 525 Bet7days 1.23 5.50 10.00 1.23 5.50 10.00
74.26 ... 叙利亚 94 4 0 0 2010-03-03 3 100.0 100.0
100.0
30 240228 90005 gavg 1.19 5.42 11.14 1.18 5.53 12.05
75.07 ... 叙利亚 94 4 0 0 2010-03-03 3 99.0 108.0
102.0
31 240228 90009 gmax 1.25 7.20 18.00 1.23 7.00 18.50
80.60 ... 叙利亚 94 4 0 0 2010-03-03 3 98.0 103.0
97.0
32 240228 90001 gmin 1.11 4.10 5.90 1.11 4.10 8.74
69.46 ... 叙利亚 94 4 0 0 2010-03-03 3 100.0 148.0
100.0

```

```
[5 rows x 38 columns]
```

请注意，最后几列数据 `kpwin`、`kplost` 和 `kpdraw` 都是预处理函数完成的。

此外，`df` 和 `xtfb.xdat10` 两个不同变量输出的结果是一样的，这是因为在 Python 程序和 Pandas 数据分析模块中，通常的赋值只是针对数据指针赋值，不是真正的变量赋值，类似某些编程语言当中的别名。

所以，虽然在 `sta310_pre` 数据预处理函数当中，没有采用“显式”对变量 `xtfb.xdat10` 进行修改，所有的修改代码都是基于变量 `df`，但最终的结果也在变量 `xtfb.xdat10` 中得到体现。

这种模式虽然在大部分时间都对提高效率有很大的帮助，但个别情况下也会引起混乱，引发不必要的错误。

这一点，特别是对于从其他编程语言迁移过来的程序员，需要多加注意。

9.4.2 案例 9-8：策略 310 参数寻优

以往的案例，我们都是根据经验设置经验参数，再运行策略寻优函数，本节 `sta310` 策略函数，我们采用更加贴近实盘的模式，先通过参数寻优寻找最佳参数，再进行实盘分析。

在实盘中，通常会有不少的现成策略可供参考，如金融量化、股票投资中的均线策略、MACD 策略、海龟策略，都是非常成熟的投资策略，不过不同的市场、不同的时间周期，同样的策略函数使用的参数是不同的，这时就需要先进行回溯程序，再寻找最佳参数。

参数寻优的一个缺点就是，回溯都是基于历史数据的，很容易造成参数的过度优化，原本测试回报率最佳的参数，用于实盘反而出现亏损。

所以在运行参数寻优时，要牢记大盘原则，即在合理的回报率下，最后结果的基数越大越好，不要单一追求最佳回报率，比回报率更加重要的是稳定性。

案例 9-8 的文件名是 `zc908_sta310var.py`，核心代码如下：

```
def main_var1x():
    #1
    mv9=pd.DataFrame(columns=tfsys.btvarSgn)
    flog='tmp/log_sta310.csv'
    vllst=[2,3,4,5,7,9]
```

```

nlst=[500,1000,2000,3000]
#
for nc in nlst:
    for vl in vllst:
        tim0=arrow.now()
        #
        xtfb=main_var100([vl],nday=nc)
        tn=zt.timNSec(' ',tim0)
        r1=xtfb.poolRet.tail(1)
        r1['v1'],r1['nday']=vl,nc
        r1['v2'],r1['v3'],r1['v4'],r1['v5']=0,0,0,0
        r1['doc']=str(round(tn))
        #
        mv9=mv9.append(r1,ignore_index=True)
        mv9.to_csv(flog,index=False,encoding='gbk')
    print(mv9)

```

sta310 策略虽然有些啰嗦，但还是一元参数的策略模型，所以我们采用的是一元的测试模板。

测试参数如下：

```

vllst=[1,3,5,7,9]
nlst=[500,1000,2000,3000]

```

此外，在测试主入口函数 `main_var100` 中，要设置好数据预处理函数和策略函数：

```

xtfb.funPre=tfsty.sta310_pre
xtfb.funSta=tfsty.sta310_sta

```

程序很简单，为了保持一致的学习体验，我们设置了 `timStr='2017-02-10'`，以统一测试周期。

此外，为了体现回报率数据，采用的是 `nday=50` 天的测试周期，而不是采用默认的 `nday=2` 天。

在 `main_bt` 回溯主函数中，本案例与以往的案例大致相同，不同的是第 3 组代码的回溯参数设置：

```

xtfb.funPre=tfsty.sta310_pre
xtfb.funSta=tfsty.sta310_sta
xtfb.preVars=[]
xtfb.staVars=vllst

```


以往的案例大部分无需进行数据预处理，只是本案例增加了数据预处理函数。

9.4.3 案例 9-9：策略 310 图表分析

案例 9-9 使用的数据文件名是 dat/log_sta310.csv，是由案例 9-8 的策略 sta310 生成的，先使用最常用的数据分析软件 Excel 查看一下，如图 9-10 所示是选择 kret 总回报率排序后的部分截图。

J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	
kret9	nday	num0	num1	num3	num9	nwin0	nwin1	nwin3	nwin9	ret0	ret1	ret3	ret9	v1	v2
93.01	1000	3532	1229	8204	12965	1717	381	4706	6804	3277.25	1218.68	7563.03	12058.96	2	
92.99	1000	3928	1687	8500	14115	1859	518	4826	7203	3649.95	1670.46	7805.31	13125.72	3	
92.99	1000	3929	1687	8501	14117	1859	518	4827	7204	3649.95	1670.46	7806.76	13127.17	4	
92.77	2000	4877	2021	10549	17447	2342	612	5973	8927	4599.57	1978.63	9608.17	16186.37	3	
92.77	2000	4878	2021	10550	17449	2342	612	5974	8928	4599.57	1978.63	9609.62	16187.82	4	
92.77	3000	4877	2021	10549	17447	2342	612	5973	8927	4599.57	1978.63	9608.17	16186.37	3	
92.77	3000	4878	2021	10550	17449	2342	612	5974	8928	4599.57	1978.63	9609.62	16187.82	4	
92.75	2000	4373	1474	10142	15989	2165	448	5788	8401	4131.1	1437.08	9262.12	14830.3	2	
92.75	3000	4373	1474	10142	15989	2165	448	5788	8401	4131.1	1437.08	9262.12	14830.3	2	
92.61	1000	4270	2181	8995	15146	1983	650	4914	7547	3956.28	2084.17	7986.51	14026.96	5	
92.58	500	2180	946	4593	7721	1016	293	2623	3932	1968.22	930.37	4249.46	7148.05	4	
92.57	500	2180	946	4593	7721	1016	293	2623	3932	1968.22	930.37	4249.46	7147.67	3	
92.25	2000	5297	2611	10807	18715	2498	772	6074	9344	4970.65	2473.12	9821.61	17265.38	5	
92.25	3000	5297	2611	10807	18715	2498	772	6074	9344	4970.65	2473.12	9821.61	17265.38	5	
92.16	500	2331	1207	4687	8225	1064	365	2659	4088	2086.07	1165.47	4328.46	7580	5	
92	500	1985	713	4462	7160	951	213	2562	3726	1798.2	673.33	4115.99	6587.52	2	
91.84	1000	4516	2645	8750	15911	2040	767	4936	7743	4115.65	2464.3	8032.21	14612.16	7	
91.72	500	2442	1427	4686	8555	1087	425	2662	4174	2143.12	1362.84	4341.08	7847.04	7	
91.69	2000	5641	3162	10876	19679	2585	922	6095	9602	5222.27	2959.09	9861.74	18043.1	7	
91.69	3000	5641	3162	10876	19679	2585	922	6095	9602	5222.27	2959.09	9861.74	18043.1	7	

图 9-10 数据文件截图

由图 9-10 可以看出：

- 与前面讲的其他案例不同，策略 sta310 的整体回报率非常稳定，都在 91.7%~93%之间；
- num3、num1、num0 的基数也非常庞大，说明 sta310 属于非常稳定的策略；
- 因为截图尺寸问题，没有胜率数据，knum3 约 57%、knum1 约 30%、knum0 约 47%，读者可以通过 Excel 查看；
- 整体来说，参数 v1 取值 2、3、4 时，回报率都差不多；
- 参考 nday 测试周期，可以看到 v1 取 2、3、4 三个数值时，在 500、1000、2000、3000 天的测试周期中 kret9 总回报率都差不多，均稳定在前面几位。

综上所述，2、3、4 三个数值是策略 sta310 的最佳参数，获得的回报率最稳定，而且获利率也处于相对高位。

案例 9-9 的文件名是 zc909_sta310vdr.py，是根据案例 9-8 sta310 一号策略的参数自动寻优程序输出的测试数据文件，再进行的二次图表分析，核心代码如下：

```
#1
fss='dat/log_sta310.csv'
df=pd.read_csv(fss,index_col=False,encoding='gbk')

#2
vlst=[2,3,4,5,7]
nlst=[500,1000,2000,3000]

df['v1']=df['v1'].astype(float)
df=df.set_index(['nday'])
df5=pd.DataFrame()

#3
for v1 in vlst:
    df2=df[df['v1']==v1]
    #print(df2)
    xss='v1_{0:.2f}'.format(v1)
    df5[xss]=df2['kret9']

#4
print(df5)
df5.plot(grid=True)
```

案例 9-9 运行流程如下。

- 第 1 组程序，读入测试数据文件到 df 变量。
- 第 2 组程序，根据测试参数列表和 Excel 对结果数据的简单分析，做一个简化的 vlst 变量列表，用来设置其他参数，并定义一个 df5 的 Pandas 矩阵 DataFrame 变量。
- 第 3 组程序，基于 vlst 的循环提取各组参数的总回报数据并保存到 df5。
- 第 4 组程序，输出 df5 的结果并绘制相关的图形。

案例 9-9 运行结果如下：

	v1_2.00	v1_3.00	v1_4.00	v1_5.00	v1_7.00
nday					
500.0	92.00	92.57	92.58	92.16	91.72
1000.0	93.01	92.99	92.99	92.61	91.84
2000.0	92.75	92.77	92.77	92.25	91.69
3000.0	92.75	92.77	92.77	92.25	91.69

输出的总回报率曲线图如图 9-11 所示。

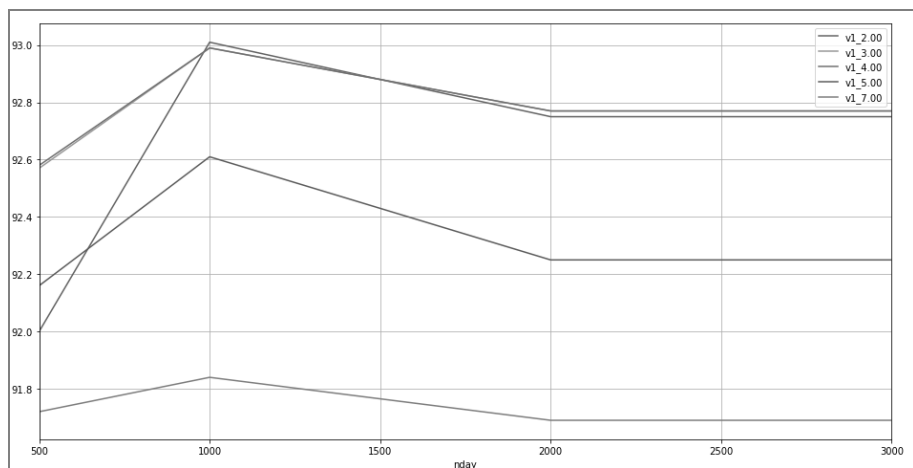


图 9-11 总回报率曲线图

由图 9-11 可以看出，除了各种参数，在测试周期 $nday$ 大于 1000 天后，都是趋于稳定的，整体回报率在 91.6-92.8%。其中， $v1$ 值取 7 时，回报率最低； $v1$ 值取 2、3、4 三个数值时，回报率都差不多，位于高位，而且非常稳定，这与我们在 Excel 报表中初步分析的结果相同。

对于出现在循环中的代码：

```
df5[xss]=df2['kret9']
```

稍作修改，就可绘制胜盘回报率的曲线图形：

```
df5[xss]=df2['kret3']
```

以及平均回报率的曲线图形：

```
df5[xss]=df2['kret1']
```

需要注意的是，本书第一次出现平局数据的图表分析，非常期待最后的曲线结果。

负盘回报率的曲线图形，代码修改如下：

```
df5[xss]=df2['kret0']
```

以下是胜盘数据的输出信息：

	v1_2.00	v1_3.00	v1_4.00	v1_5.00	v1_7.00
nday					
500.0	92.25	92.51	92.52	92.35	92.64
1000.0	92.19	91.83	91.83	91.85	91.80
2000.0	91.32	91.08	91.09	90.88	90.67
3000.0	91.32	91.08	91.09	90.88	90.67

如图 9-12 所示是胜盘回报率的曲线图。

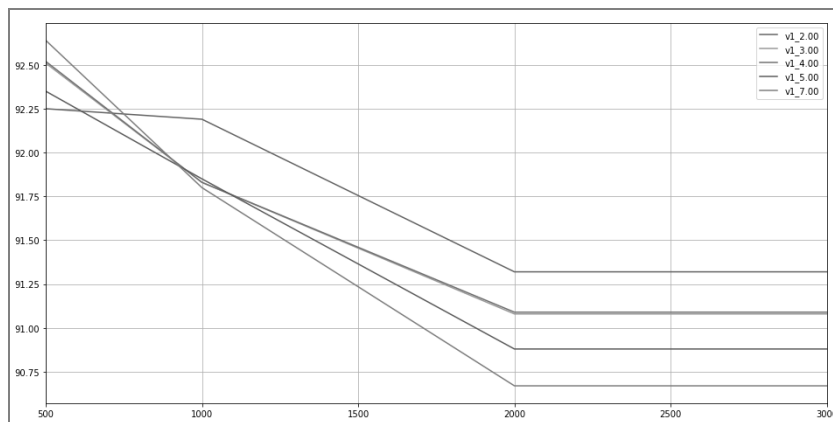


图 9-12 胜盘回报率曲线图

由图 9-12 可以看出，在胜盘曲线中，参数 $v1$ 值取 2 时，回报率最高。以下是平局数据的输出信息：

	$v1_2.00$	$v1_3.00$	$v1_4.00$	$v1_5.00$	$v1_7.00$
nday					
500.0	94.44	98.14	98.14	96.56	95.50
1000.0	99.16	99.02	99.02	95.56	93.17
2000.0	97.50	97.90	97.90	94.72	93.58
3000.0	97.50	97.90	97.90	94.72	93.58

如图 9-13 所示是平局回报率的曲线图。

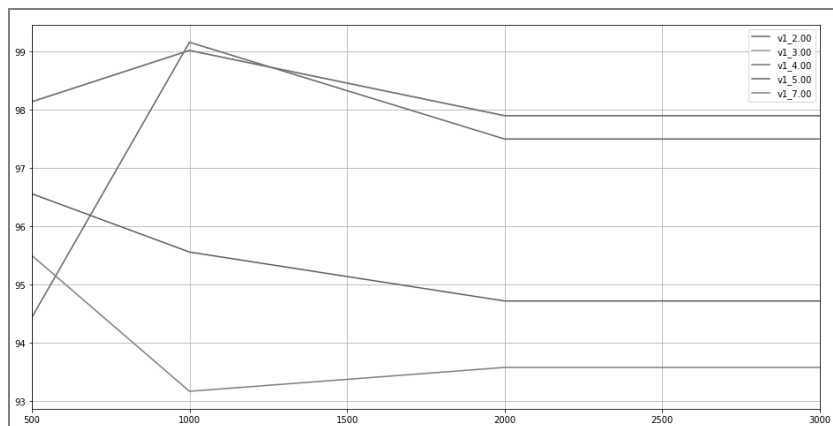


图 9-13 平局回报率曲线图

这是我们第一次分析平局数据的回报率收益曲线，由图 9-13 可以看出，在平局的回报率曲线中，当参数 $v1$ 值取 2 和 4 时，回报率最高。

以下是负盘数据的输出信息：

	$v1_2.00$	$v1_3.00$	$v1_4.00$	$v1_5.00$	$v1_7.00$
nday					
500.0	90.59	90.29	90.29	89.49	87.76
1000.0	92.79	92.92	92.90	92.65	91.13
2000.0	94.47	94.31	94.29	93.84	92.58
3000.0	94.47	94.31	94.29	93.84	92.58

如图 9-14 所示是负盘回报率的曲线图。

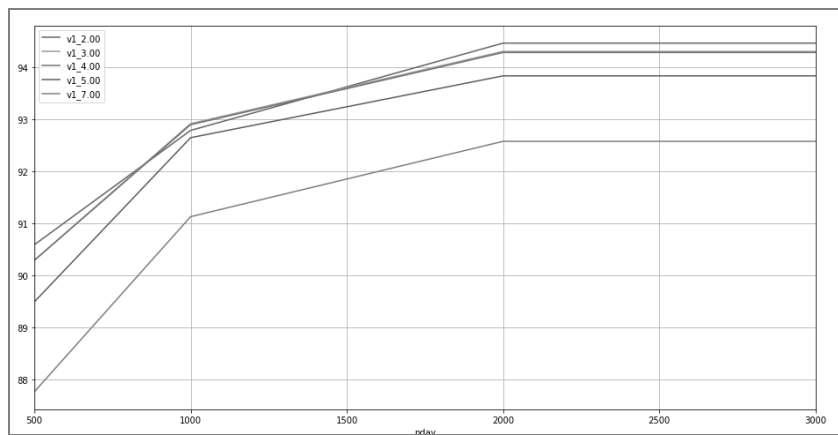


图 9-14 负盘回报率曲线图

由图 9-14 可以看出，在负盘比赛中，曲线走势稳定，回报率最高的是参数 2。因此，实盘中负盘如果采用 2 作为赔率参数，理论上总体回报率更高。

在 $kret9$ 的分析中，参数 $v1$ 值取 2、3、4 这三个数值时，回报率都差不多，均位于高位，而且非常稳定。在 $kret3$ 胜盘回报率曲线图中， $v1$ 值取 2 时，有明显的优势。其他平局、负盘的回报率曲线图中， $v1$ 值取 2 时，也是收益率居于前。因此，最终的分析结果表明， $v1$ 值取 2 是策略 $sta310$ 的最佳参数。

遗憾的是，数值 2 是本次参数寻优的下限数值，在实盘中，应该再加入数据 0 和 1，甚至 -1、-2 等数据，重新进行寻优测试。

此外，读者也可以参考一号策略扩展的介绍与源码，对 $sta310$ 策略进行优化和调整，这些留给读者自己测试。

9.4.4 案例 9-10: 策略 310

案例 9-10 的文件名是 `zc910_sta310.py`, 核心代码如下:

```
timStr='2017-02-10'
main_bt(timStr,50)
print('\nok!')
```

程序很简单, 为了保持一致的学习体验, 我们设置了 `timStr='2017-02-10'`, 以统一测试周期。

此外, 为了体现回报率数据, 采用的是 `nday=50` 天的测试周期, 而没有采用默认的 `nday=2` 天。

在 `main_bt` 回溯主函数中, 与以往的案例大致相同, 不同的是第 3 组代码的回溯参数设置:

```
xtfb.funPre=tfsty.sta310_pre
xtfb.funSta=tfsty.sta310_sta
xtfb.preVars=[]
xtfb.staVars=[2]
```

以往的案例大部分无需进行数据预处理, 策略 310 增加了数据预处理函数, 策略参数采用前面测试的结果, 数值为 2。

案例 9-10 的部分运行结果如下:

```
#4,result.anz

xtfb.poolTrd,足彩推荐
```

tweek	gid	gset	mplay	mtid	gplay	gtid	qj	qs	qr	kend	kwin	kwinrq
		tplay		tsell	cid	pwin9	pdraw9	plost9		kwin_sta		
0	581454	澳超	纽喷射	3617	墨尔本	3615	-1.0	-1.0	0.0	0	-1.0	
-1.0	1	2017-02-13	2017-02-13	16:45:00	1	3.65	3.75	1.70	0.0			
1	610611	意甲	拉齐奥	964	AC米兰	210	-1.0	-1.0	0.0	0	-1.0	
-1.0	1	2017-02-13	2017-02-13	23:55:00	1	1.62	3.55	4.35	3.0			
2	607051	西甲	埃瓦尔	562	格拉纳	4607	-1.0	-1.0	0.0	0	-1.0	
-1.0	1	2017-02-13	2017-02-13	23:55:00	1	1.43	3.85	5.90	3.0			
3	572965	英超	伯恩茅	667	曼城	1072	-1.0	-1.0	0.0	0	-1.0	
-1.0	1	2017-02-13	2017-02-13	23:55:00	1	6.60	4.70	1.31	0.0			
4	574023	法甲	埃蒂安	494	洛里昂	980	4.0	0.0	0.0	1	3.0	
-1.0	0	2017-02-13	2017-02-12	23:55:00	1	1.51	3.57	5.35	3.0			

```

xtfb.poolRet, 回报率汇总
      xtim  kret9  kret3  kret1  kret0  knum9  knum3  knum1
knum0  ret9 ...   ret3  ret1  ret0  nwin3  nwin1  nwin0  num3  num1
num0  cid
    40 2016-12-27  57.67  86.50   0.00   0.00  50.00  75.00   0.00
0.00  3.46 ...   3.46  0.00   0.00   3.0   0.0   0.0   4.0   2.0
0.0   1
    41 2016-12-26  61.63  83.64   0.00   0.00  42.11  57.14   0.00
0.00 11.71 ...  11.71  0.00   0.00   8.0   0.0   0.0  14.0   4.0
1.0   1
    42 2016-12-24 115.50 116.00   0.00 115.00  75.00  75.00   0.00
75.00  9.24 ...   4.64  0.00   4.60   3.0   0.0   3.0   4.0   0.0
4.0   1
    43 2016-12-23  85.78  65.29 315.00   0.00  44.44  42.86 100.00
0.00  7.72 ...   4.57  3.15   0.00   3.0   1.0   0.0   7.0   1.0
1.0   1
    44      sum  88.15  87.53  98.77  82.92  49.79  58.21  30.86
44.36 424.86 ... 234.57 80.00 110.29 156.0  25.0  59.0 268.0 81.0
133.0  1

[5 rows x 22 columns]
kcid, 1 ,nday, 50
preVar, []
staVar, [2]

#5,backtest,tim:26.66 s

```

由运行结果可以看到，胜盘的回报率是 88%，负盘的回报率都在 80% 多，而平局的回报率有些高 98.77%，不过 num1=81，平局的比赛场次也不少，结果也属于合理范围。

10

第 10 章

Python 人工智能入门与实践

10.1 从忘却开始

广大初学者面对人工智能、机器学习这些高大上的概念，一方面迫切希望能够掌握相关的知识，另一方面，面对各种层出不穷的概念，往往会眼花缭乱、无所适从。

目前 Python 已经是人工智能、机器学习的行业标准语言，TensorFlow、Torch 于 2015 年、2016 年先后开放了 Python 语言接口。

Sklearn 是 Python 语言最重要的人工智能模块库，目前已经收入 Scikit 套件，所以又称为 scikit-learn，不过通常还是简称为 Sklearn。

如图 10-1 所示是 Sklearn 官方网站的人工智能、机器学习知识图谱。

不知道读者对于此图的第一感觉是什么，反正笔者看到这幅图的第一反应是吓呆了，也明白了为什么这么多初学者对于人工智能望而生畏。

人工智能、机器学习、数据分析是笔者研究了多年的课题与领域，20 年前笔者最早的论文标题就是《人工智能与中文字型设计》，该论文还在与暴雪（你没看错，就是做魔兽的暴雪）、宝洁等国际 500 强的知识产权案件中，被作为技术文献引用。

既然笔者的论文能够用于非常严肃的司法领域，那么在人工智能领域，笔者也可以算是根正苗红的学者了。

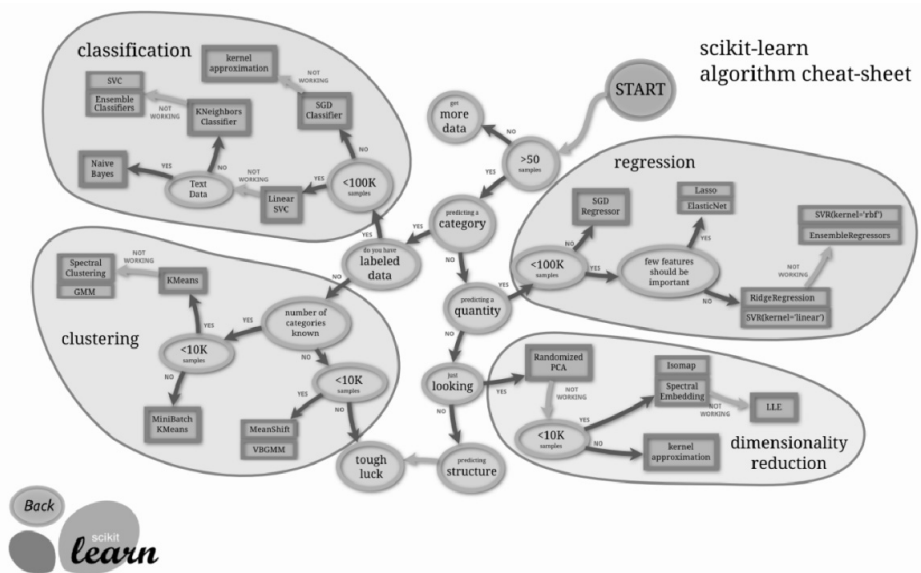


图 10-1 人工智能、机器学习知识图谱

与普通理论学者不同的是，笔者始终认为自己是一名软件工程师或者程序员，也就是网络上常说的“码农”。

对于程序员而言：Talk is cheap, show me the code!

再多的理论也比不上几件成功的软件作品，笔者虽然谈不上有很多成功的软件作品，但也写过不少专业的程序，例如原生的 OCR 识别程序和英汉翻译程序。

所谓原生程序，就是直接采用 C 语言、Delphi 语言的标准函数库，没有采用任何第三方 AI 架构库，如使用 Sklearn、TensorFlow 等 AI 模块库直接编程，做过系统的程序员或者阅读 Linux、安卓系统、TensorFlow 系统源码的程序员，会明白期间的难度。

这种基于原生的开发，最大的好处是，无论采用何种编程语言、何种理论算法，最终的底层代码结构都差不多，就像电脑里面的 CPU，算得再快也不过是个加法器。

笔者曾经在博客《文科生、易经与大数据》中说过：

殊途同归，什么东西到了极致，其根源都是相通的。

易经是纯文科的了，字王小数据理论、“黑天鹅算法”，灵感就是来自：易经、阴阳、八卦……

以量化投资为例，在进行大数据分析时，我们发现，所有的分析，抛开表象，到了最后，无非是两种选择：亏、盈。

延伸一下，其他项目也是如此：

输，赢；正，负；胜，负；涨，跌；加，减；男，女；老，少；黑，白；取，舍。

这些正好对应易经的：阴、阳。

计算机的核心是 CPU，但 CPU 的本质不过是一个“加法器”。

在 CPU 里面，其实加减都是加法，因为减不过是加上一个负数符号。

这个加法器正好对应了易经里面的一生二：一（CPU 加法器）生二（加、减）。

明白了这点再看 Sklearn 的知识图谱，虽然表面看起来还是乱七八糟、非常烦琐，但仔细梳理后能够看出其核心只有一个词：Type，即分类。

将分类做好，其他匹配、识别都是简单的问题，只是贴上标签和加上备注的问题。

国外也有专家认为，所有的人工智能、机器学习，本质上都是二元一次方程的寻优算法。

在笔者的博客中曾经也说过：

可以把人工智能、机器学习看成一个巨大的字符串 Find 查找算法，只不过这个算法中的 keyword（关键词）与被查找的字符串的大小非常庞大，趋于无限长度。

众所周知，开源项目 70% 的问题都是卡在系统配置上，特别是国外的项目，有语言障碍、文档不齐。

TopQuant.vip 极宽量化开源社区发布的集成式 Python 开发平台：zwPython，解压即用，无须安装；内置各种人工智能、机器学习所需的各种模块库，包括 NLTK、Sklearn，以及最高大上的 TensorFlow，而且全功能、全免费、全开源，无须花费一分钱，是真正的零起点学 AI。

从某种程度上而言，能够下载 zwPython，解压并且运行内置的 hello 程序，相对于传统的人工智能学习周期，就已经完成了 70%，已经拥有了专业级别的人工智能开发平台。

很多初学者，面对人工智能、机器学习往往连系统配置都不会，最基本的 hello 程序也无法运行。

在“zwPython 用户手册”里面，笔者曾经说过，虽然很多人认为 Python 是面向对象的语言，但实践表明，忘记 OOP 对象编程的概念，采用传统的 BASIC 语言（面向过程）模式，学习效率可以提高 10 倍以上。

这一点用过 zwPython 的用户都知道,笔者量化培训班的学员更是深有体会,许多 40 多岁金融一线的从业人员完全没有编程基础,通常学习两个月,就可以自己修改 zwQuant 量化开源软件中的策略函数。

所以,学习人工智能,笔者的建议就是:从忘却开始,忘却各种乱七八糟的概念。

忘记这些概念之后,读者就会觉得海阔天空,所谓的人工智能,不过是传统的 Python 函数调用,而且只有寥寥无几的数个函数,在最简单的案例当中,只需要 2~3 个 API 相关的函数。

需要说明的是,对于初学者而言,除了忘却之外,不要左顾右盼, MATLAB、R 语言、Torch、TensorFlow、NLTK……这些都想学,但或许都学不好。

以上这些名词术语看起来简单,但其实每一个名词术语,若没有一个专业的博士团队,都是无法完成的独立项目。

对于初学者而言,入门阶段就学习一个模块库: Sklearn。

Sklearn 模块库本身就是人工智能、机器学习的行业标准,该有的人工智能、机器学习经典算法全部都有,其他的模块库无非是在局部进行了某些优化。

至于人工智能的进阶课程,读者不要着急,学习完 Sklearn 的人工智能入门课程后,再看 TensorFlow、Torch 就不会有看天书的感觉了。

10.2 Iris 经典爱丽丝

Iris 爱丽丝是鸢尾属(拉丁学名: Iris L.)的单叶植物,与常见的百合花类似。

爱丽丝数据集是人工智能、机器学习最经典的数据集,全称是 Fisher's Iris Data (安德森鸢尾花卉数据集),是统计学习的必备数据集。

维基百科有专门的词条:

安德森鸢尾花卉数据集(英文: Anderson's Iris data set),也称鸢尾花卉数据集(英文: Iris flower data set)或费雪鸢尾花卉数据集(英文: Fisher's Iris data set),是一类多重变量分析的数据集。它最初是埃德加·安德森集从加拿大加斯帕半岛上的鸢尾属花朵中提取的地理变异数据,后由罗纳德·费雪作为判别分析的一个例子,并运用到统计学中。其数据集包含了 50 个样本,都属于鸢尾属下的 3 个亚属,分别是山鸢尾、变色鸢尾和维吉尼亚鸢尾。其 4 个特征被用做样本的定量分析,即花萼和花瓣的长度和宽度。基于这 4 个特征的集合,费雪发展了线性判别分析以确定其属种。

我们的目的就是通过编程对这三种不同种类、150 多组爱丽丝植物的数据，采用专业的数据分析手段和人工智能算法让程序自动判断植物的种类。

10.2.1 案例 10-1：经典爱丽丝

从本章开始讲解人工智能，我们的课件程序编码使用 **zai** 开头，其中 **ai** 是人工智能的意思。

案例 10-1 的文件名是 `zai101_iris01.py`，核心代码如下：

```
#1
fss='dat/iris.csv'
df=pd.read_csv(fss,index_col=False)
print('\n#1 df')
print(df.tail())
print(df.describe())

#2
d10=df['xname'].value_counts()
print('\n#2 xname')
print(d10)
```

第 1 组运行结果如下：

	x1	x2	x3	x4	name
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

	x1	x2	x3	x4
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000

```
max      7.900000    4.400000    6.900000    2.500000
```

第 2 组代码中的函数如下：

```
dl0=df['xname'].value_counts()
```

对于植物种类进行了简单的分类统计，共有 3 种，对应的输出信息如下：

```
#2 xname
versicolor    50
setosa         50
virginica      50
Name: xname, dtype: int64
```

由输出信息可以看出，3 种植物名称分别是山鸢尾（*Iris setosa*）、变色鸢尾（*Iris versicolor*）和维吉尼亚鸢尾（*Iris virginica*）。

很多人看完以上程序可能觉得有些过于简单，案例 10-1 作为人工智能的入门程序的确有些过于简单，但很多初学者就卡在这简简单单的入门案例中。

看起来案例 10-1 似乎很简单，但其实做了很多小优化和修改：取消了原始的数据列名称，用 x1~x4 代替，与 Top-AI 极宽的 AI 模块兼容；另外，更加符合人工智能、机器学习的本质，对于机器学习而言，无所谓采用什么名称，在内部都是数组矩阵。

此外，从输出信息来看，统计命令如下：

```
print(df.describe())
```

在对应的输出信息中没有 xname 数据，因为 xname 字段是字符串，无法统计分析，需要先对其进行数字化处理，也就是常说的文字信息的矢量化运算。

全程采用 Pandas 学习 Sklearn 人工智能也可以说是笔者的一个微创新，方便初学者把握数据内部的构成。

传统的 Sklearn 人工智能文档大部分直接采用 Numpy 数组，而 Numpy 是为了追求极限性能设计的模块库，很多算法函数非常复杂，不亚于汇编语言。

从某种程度上而言，绝大部分初学者的人工智能学习之路，起步阶段就被 Numpy 这个模块库给吓退了。

虽然 GitHub 项目的 Pandas-Sklearn 项目已经启动，但这个项目为了提高效率采用了内部耦合，对于初学者帮助不大。

本书全部采用现有的 Pandas 命令，从数据源对 Sklearn 进行整合，无须学习额外的语法，更加方便初学者入门。

10.2.2 案例 10-2：爱丽丝进化与矢量化文本

案例 10-1 有一个小 Bug，即统计命令：

```
print(df.describe())
```

在对应的输出信息中没有 `xname` 数据，因为 `xname` 字段是字符串，无法统计分析，需要先对其进行数字化处理，也就是常说的文字信息的矢量化运算。

案例 10-2 的文件名是 `zai102_iris02.py`，将根据 `xname` 植物名称设置一个新的数据字段 `xid` 来完成这个文本信息的矢量化工作。

案例 10-2 很简单，下面分组逐一讲解。

第 1 组代码，读取 Iris 数据文件，并保存到 `df` 变量：

```
#1
fss='dat/iris.csv'
df=pd.read_csv(fss,index_col=False)
```

第 2 组代码，根据 `xname` 字段，按 1、2、3 分别设置 `xid` 字段，完成读取爱丽丝数据名称的矢量化操作，`xid` 数据字段格式设置为 `int` 整数格式，并保存到文件 `iris2.csv`。

```
#2
df.loc[df['xname']=='virginica', 'xid'] = 1
df.loc[df['xname']=='setosa', 'xid'] = 2
df.loc[df['xname']=='versicolor', 'xid'] = 3
df['xid']=df['xid'].astype(int)
df.to_csv('tmp/iris2.csv',index=False)
```

我们已经将 `iris2.csv` 文件复制到 `dat` 目录，在后面的案例中，读者可以直接使用这个文件作为数据源：`dat/iris2.csv`。

第 3 组代码，输出修改后的 `df` 数据信息：

```
#3
print('\n3#df')
print(df.tail())
print(df.describe())
```

对应的输出信息是：

	x1	x2	x3	x4	xname	xid
145	6.7	3.0	5.2	2.3	virginica	1
146	6.3	2.5	5.0	1.9	virginica	1
147	6.5	3.0	5.2	2.0	virginica	1
148	6.2	3.4	5.4	2.3	virginica	1

```

149  5.9  3.0  5.1  1.8  virginica  1
      x1      x2      x3      x4      xid
count 150.000000 150.000000 150.000000 150.000000 150.000000
mean   5.843333  3.054000  3.758667  1.198667  2.000000
std    0.828066  0.433594  1.764420  0.763161  0.819232
min    4.300000  2.000000  1.000000  0.100000  1.000000
25%    5.100000  2.800000  1.600000  0.300000  1.000000
50%    5.800000  3.000000  4.350000  1.300000  2.000000
75%    6.400000  3.300000  5.100000  1.800000  3.000000
max    7.900000  4.400000  6.900000  2.500000  3.000000

```

第 4 组代码，输出 `xname` 方面的分类统计信息：

```

d10=df['xname'].value_counts()
print('\n3#xname')
print(d10)

```

对应的输出信息是：

```

4#xname
versicolor      50
setosa           50
virginica        50
Name: xname, dtype: int64

```

第 5 组代码，输出 `xid` 方面的分类统计信息：

```

d10=df['xid'].value_counts()
print('\n4#xid')
print(d10)

```

对应的输出信息是：

```

4#xid
3      50
2      50
1      50
Name: xid, dtype: int64

```

10.3 AI操作流程

本节涉及部分人工智能的理论介绍，某些内容有些抽象拗口，可能读者感觉无法细分，属于正常情况，在后面的案例中会具体介绍，读者无须纠结。

10.3.1 机器学习与测试数据集

人工智能、机器学习通常都使用两组数据，一组作为训练数据，另外一组作为测试数据。

在每组数据中都包含一组多维的参数数据集作为特征数据集，以及一组一维的数组作为结果分类数据，从而形成 4 个数据集，通常这 4 组数据变量的名称如下。

- `x_train`，训练数据，多维参数数据集。
- `y_train`，训练数据，一维结果数据集。
- `x_test`，测试数据，多维参数数据集。
- `y_test`，测试数据，一维参数数据集。

习惯上，`train` 数据集用于训练，`test` 数据集用于测试，此外通过对数据集 `y_train` 的分析，会生成一个新的预测结果数据集 `y_pred`，这个数据集也是一维的结果数据集。

通过对结果数据集 `y_pred` 与实际的测试数据集 `y_test` 进行对比，就可以检测算法模型的准确度。

10.3.2 机器学习运行流程

通常人工智能、机器学习的算法流程如下。

- 选择模型函数 `mx_fun`，`mx_fun` 是我们的自定义机器学习函数接口。
- 把训练用的特征数据集 `x_train` 和对应的特征或者结果数据集 `y_train`，输入模型函数 `mx_fun`。
- 系统内置的机器学习函数，会自动分析特征数据与结果数据之间的关系，这个过程就是机器学习的过程，也可以说是算法建模的过程。
- 通过对训练数据的机器学习和数据分析，系统会生成一个 AI 机器学习的模型，我们将其保存到变量 `mx`。
- 把测试数据 `x_test` 输入 `mx` 模型变量，`mx` 会调用内置的分析函数 `predict`，生成最终的分析结果 `y_pred`。
- 如果是实盘，输入最新的数据，例如今天的股市数据、正在销售的足彩比赛赔率数据，系统会自动生成相关的预测数据，例如每天或未来几天股市的走势数据、比赛输赢结果。

在进行实盘运行前,对预测结果数据 `y_pred` 和正确的结果数据 `y_test` 进行对比,以判断模型的准确程度,并通过一些优化措施和结果调整参数进行迭代运算,或者采用其他的模型,以提高最终结果的准确度。

以下是常用的 AI 运算流程:

选择模型函数 `mx_fun` → 导入训练数据 → 建立算法模型 `MX` → 输入测试(实盘)数据 `x_test` → 调用 `predict` 分析(预测)函数 → 生成分析(预测)结果 `y_pred`

10.3.3 经典机器学习算法

目前,人工智能、机器学习虽然光彩夺目,但还处于启蒙阶段,在本书的案例中,涉及了以下最为经典的机器学习算法。

- 线性回归,函数名为 `LinearRegression`。
- 朴素贝叶斯算法,函数名为 `Multinomialnb`。
- KNN 近邻算法,函数名为 `KNeighborsClassifier`。
- 逻辑回归算法,函数名为 `LogisticRegression`。
- 随机森林算法,函数名为 `RandomForestClassifier`。
- 决策树算法,函数名为 `tree.DecisionTreeClassifier`。
- GBDT 迭代决策树算法,又叫 MART (Multiple Additive Regression Tree),函数名为 `GradientBoostingClassifier`。
- SVM 向量机算法,函数名为 `SVC`。
- SVM- cross 向量机交叉算法,函数名为 `SVC`。

以上算法中的函数名均为 Sklearn 模块库内置的函数名称,无须使用其他第三方模块库,所以笔者认为,Sklearn 模块库是初学者学习人工智能、机器学习的不二选择。

10.3.4 黑箱大法

在本章开头笔者说过,部分理论内容有些抽象拗口,如果读者感觉无法理解,也属于正常情况,后面的案例会具体介绍,目前无须纠结。

这些还只是人工智能、机器学习最简单的部分。大部分初学者，即使克服重重困难，独自完成了人工智能、机器学习开发平台的配置，再面对这些拗口的算法名称，也会有崩溃的感觉。

其实这种现象很正常，因为这些算法、名称的背后都有非常专业的理论和模型，其学术价值和专业难度都不亚于博士学位的研究。

不过，正如笔者前面所说，初学者学习人工智能、机器学习最好从忘却开始。

同样，面对这些眼花缭乱的专业名称术语，依然采用忘却的模式，使用黑箱大法，读者无须纠结各种算法背后的理论，只将其看做是一个个黑箱函数即可：

输入数据 → 【黑箱分析】 → 获得结果

以上各种算法，我们甚至可以将其改名为 1 号函数、2 号函数、3 号函数等。

函数调用是 Python 语言的基本功，能够看到这里的读者，相信无论对函数的调用还是编程都已经非常熟悉。

市场经济讲究的是结果导向，对于大部分读者而言，需要的也只是一个最终的结果数据。

如此一来，99% 的人工智能理论知识都可以只简单了解，然后直接调用相关的函数，获取最后的结果。

人工智能、机器学习领域的专家学者，也可以采用这种黑箱模式，有了结果数据，再学习理论和算法，就会有具体的数据支持和更多的感性认识，研究过程也会事半功倍。

10.3.5 数据切割函数

稍后的案例 10-3 会涉及第一个 Sklearn 的专业函数：

```
train_test_split
```

`train_test_split` 函数只是对数据进行切割，属于数据预处理函数，并非正式的人工智能、机器学习函数。

我们也可以通过其他函数或者自行编写的函数完成类似的功能，在前面的案例中，数据文件的切割也有类似的功能，事实上，稍后的足彩数据机器学习建模使用的数据源，正是自定义数据文件切割函数。

对于普通的小型数据集而言，Sklear 内置的 `train_test_split` 函数使用更加方便。

`train_test_split` 数据分割函数位于 Sklearn 的 `cross_validation` 子模块，功能是从样本中按比例随机选取 `train data` 和 `test data`，调用格式一般为：

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4,
random_state=0)
```

其中：

- `x` 是训练参数的数据集合；
- `y` 是训练参数 `x` 对应的结果数据集合；
- `test_size` 是样本占比，如果是整数，就是样本的数量；
- `random_state` 是随机数的种子。

10.3.6 案例 10-3：爱丽丝分解

前面我们说过，人工智能、机器学习通常都使用两组数据，形成 4 个数据集合，通常这 4 组数据变量的名称如下。

- `x_train`，训练数据，多维参数数据集。
- `y_train`，训练数据，一维结果数据集。
- `x_test`，测试数据，多维参数数据集。
- `y_test`，测试数据，一维参数数据集。

习惯上 `train` 数据集用于训练，`test` 数据集用于测试，此外通过对数据集 `y_train` 的分析，会生成一个新的预测结果数据集 `y_pred`，这个数据集也是一维的结果数据集。

案例 10-3 的文件名是 `zai103_iris03.py`，具体讲解如何分割相关的数据，下面分组进行介绍。

第 1 组代码，读取 Iris 数据文件，并保存到 `df` 变量：

```
#1
fss='dat/iris2.csv'
df=pd.read_csv(fss,index_col=False)
```

请注意，这里使用的是我们修改后、增加了 `xid` 的爱丽丝数据源文件。

第 2 组代码，输出 `df` 数据信息：

```
#2
print('\n2#df')
print(df.tail())
```

对应的输出信息是：

	x1	x2	x3	x4	xname	xid
145	6.7	3.0	5.2	2.3	virginica	1
146	6.3	2.5	5.0	1.9	virginica	1
147	6.5	3.0	5.2	2.0	virginica	1
148	6.2	3.4	5.4	2.3	virginica	1
149	5.9	3.0	5.1	1.8	virginica	1

第 3 组代码，根据人工智能、机器学习算法要求，设置总的数据库源 x、y：

```
#3
xlst,ysgn=['x1','x2','x3','x4'],'xid'
x,y= df[xlst],df[ysgn]
#
print('\n3# xlst,',xlst)
print('ysgn,',ysgn)
print('x')
print(x.tail())
print('y')
print(y.tail())
```

对应的输出信息是：

```
3# xlst, ['x1', 'x2', 'x3', 'x4']
ysgn, xid
x
      x1  x2  x3  x4
145  6.7  3.0  5.2  2.3
146  6.3  2.5  5.0  1.9
147  6.5  3.0  5.2  2.0
148  6.2  3.4  5.4  2.3
149  5.9  3.0  5.1  1.8
y
145    1
146    1
147    1
148    1
149    1
Name: xid, dtype: int64
```

第 4 组代码，生成 x_train、x_test、y_train、y_test 数据，并输出相关数据的数

据格式信息:

```
#4
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=1)
x_test.index.name, y_test.index.name = 'xid', 'xid'
print('\n4# type')
print('type(x_train),', type(x_train))
print('type(x_test),', type(x_test))
print('type(y_train),', type(y_train))
print('type(y_test),', type(y_test))
```

对应的输出信息是:

```
4# type
type(x_train), <class 'pandas.core.frame.DataFrame'>
type(x_test), <class 'pandas.core.frame.DataFrame'>
type(y_train), <class 'pandas.core.series.Series'>
type(y_test), <class 'pandas.core.series.Series'>
```

第 5 组代码, 保存相关的数据:

```
#5
fs0='tmp/iris_'
print('\n5# fs0,', fs0)
x_train.to_csv(fs0+'xtrain.csv', index=False);
x_test.to_csv(fs0+'xtest.csv', index=False)
y_train.to_csv(fs0+'ytrain.csv', index=False, header=True)
y_test.to_csv(fs0+'ytest.csv', index=False, header=True)
```

需要注意的是以下代码:

```
y_test.to_csv(fs0+'ytest.csv', index=False, header=True)
```

其中的 `header` 参数我们很少使用, 强制保存 `xid` 字段头信息, 不然 `y` 数据集会缺少字段头 `xid`, 少一行数据, 与 `x` 数据集尺寸不匹配, 读者可以自己测试一下, 看看 `header` 值为 `False` 的结果。

对应的输出信息:

```
5# fs0, tmp/iris_
```

需要说明的是, 以上数据我们均会复制到 `dat` 目录中, 以用于稍后的案例。

第 6 组代码, 输出 `x` 数据集:

```
#6
print('\n6# x_train')
print(x_train.tail())
```

```
print('\nx_test')
print(x_test.tail())
```

对应的输出信息:

```
6# x_train
      x1  x2  x3  x4
133  6.3  2.8  5.1  1.5
137  6.4  3.1  5.5  1.8
72   6.3  2.5  4.9  1.5
140  6.7  3.1  5.6  2.4
37   4.9  3.1  1.5  0.1
```

```
x_test
      x1  x2  x3  x4
xid
128  6.4  2.8  5.6  2.1
114  5.8  2.8  5.1  2.4
48   5.3  3.7  1.5  0.2
53   5.5  2.3  4.0  1.3
28   5.2  3.4  1.4  0.2
```

第 7 组代码, 输出 y 数据集:

```
#7
print('\n7# y_train')
print(y_train.tail())
print('\ny_test')
print(y_test.tail())
```

对应的输出信息:

```
7# y_train
133    1
137    1
72     3
140    1
37     2
Name: xid, dtype: int64

y_test
xid
128    1
```

```
114    1
48    2
53    3
28    2
Name: xid, dtype: int64
```

10.3.7 案例 10-4：线性回归算法

前面我们说过，所谓的人工智能不过是传统的 Python 函数调用，而且只有寥寥无几的几个函数，在最简单的案例中只需要 2~3 个函数。

有了合适的训练数据和测试数据，人工智能真的很简单，只要两三个函数，就可完成相关的人工智能、机器学习编程。

案例 10-4 的文件名为 `zai104_iris04.py`，是一个 100% 的人工智能、机器学习程序，是人工智能中最常用的线性回归算法，通过对输入数据的学习，自动对测试数据进行分类。

百度百科对应的线性回归词条是：

线性回归是利用数理统计中的回归分析，来确定两种或两种以上变量之间相互依赖的定量关系的一种统计分析方法，运用十分广泛。其表达形式为 $y = w'x + e$ ， e 为误差服从均值为零的正态分布。

案例 10-4 采用的是线性回归算法，如图 10-2 所示。

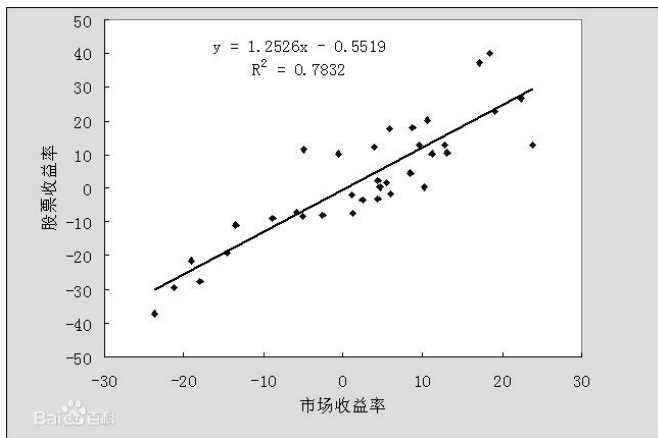


图 10-2 线性回归算法

线性回归算法是最简单、最经典、最古老的人工智能算法，其背后的理论非常复杂，在此，我们采用前面所说的黑箱模式，不予深入讨论，有兴趣的读者请自行参考相关资料。

下面逐一对程序代码进行讲解。

第 1 组代码很简单，读取训练数据，并保存到相关变量：

```
#1
fs0='dat/iris_'
print('\n1# fs0,',fs0)
x_train=pd.read_csv(fs0+'xtrain.csv',index_col=False);
y_train=pd.read_csv(fs0+'ytrain.csv',index_col=False);
```

第 2 组代码，输出部分训练数据尾部：

```
#2
print('\n2# train')
print(x_train.tail())
print(y_train.tail())
```

对应的输出信息：

```
2# train
      x1  x2  x3  x4
107  6.3  2.8  5.1  1.5
108  6.4  3.1  5.5  1.8
109  6.3  2.5  4.9  1.5
110  6.7  3.1  5.6  2.4
111  4.9  3.1  1.5  0.1
      xid
107     1
108     1
109     3
110     1
111     2
```

第 3 组代码，调用机器学习函数，通过对输入数据的分析、学习，建立机器学习的模型，并保存到变量 **mx**：

```
#3
print('\n3# 建模')
mx =zai.mx_line(x_train.values,y_train.values)
```

本案例中第 3 组代码使用的是线性回归算法建立机器学习模型，但是并没有直

接调用 Sklearn 模块库中 LinearRegression 线性回归建模函数,而是通过 ztop_ai 极宽智能模块库的 mx_line 函数接口间接进行调用,对应的函数代码是:

```
def mx_line(train_x, train_y):
    mx = LinearRegression()
    mx.fit(train_x, train_y)
    #print('\nlinreg.intercept_')
    #print (mx.intercept_);print (mx.coef_)

    return mx
```

线性回归函数位于 sklearn.linear_model 模块, 函数接口是:

```
LinearRegression(fit_intercept=True, normalize=False, copy_X= True,
n_jobs=1)
```

mx_line 函数代码很简单, 调用基于最小二乘法的 LinearRegression 线性回归函数, 生成模型变量 mx, 运行内置的 fit 命令, 分析学习训练数据: train_x (训练数据)、train_y (训练数据对应的答案)。

Sklearn 模块库中的各种机器学习函数, 基本上都是采用 fit 命令自动学习、建立模型。

采用 ztop_ai 极宽智能模块库的 mx_xxx 系列函数, 对 Sklearn 模块的各种智能算法函数进行二次封装, 其优点是:

- 统一调用接口, 规范函数 API 调用模式;
- 无须直接面对底层的机器学习函数, 如 LinearRegression, 无须了解相关的理论知识即可直接使用。

如果不考虑各种复杂的机器学习函数名称, 只需记住 mx_xxx 系列函数, 那么建模就基本简化成一个最简单的 fit 内置函数。

第 4 组代码, 读入测试数据并输出相关信息:

```
#4
x_test=pd.read_csv(fs0+'xtest.csv',index_col=False)
df9=x_test.copy()
print('\n4# x_test')
print(x_test.tail())
```

对应的输出信息是:

```
4# x_test
      x1    x2    x3    x4
33  6.4  2.8  5.6  2.1
```

```
34  5.8  2.8  5.1  2.4
35  5.3  3.7  1.5  0.2
36  5.5  2.3  4.0  1.3
37  5.2  3.4  1.4  0.2
```

第 5 组代码，运行机器学习变量 `mx` 的内置函数 `predict`，生成结果数据：

```
#5
print('\n5# 预测')
y_pred = mx.predict(x_test.values)
df9['y_preds_r']=y_pred
```

第 6 组代码，读入训练数据的正确答案并保存到变量 `y_test`：

```
#6
y_test=pd.read_csv(fs0+'ytest.csv',index_col=False)
print('\n6# y_test')
print(y_test.tail())
```

对应的输出信息是：

```
6# y_test
   xid
33    1
34    1
35    2
36    3
37    2
```

需要注意的是，本案例为了强调学习，特意把实盘数据放在 `predict` 函数之后，在生成结果或者预测数据之后再读入，以强调两者之间是完全独立的。

第 7 组代码，整理结果数据变量 `df9`，并保存到文件：

```
#7
df9['y_test'],df9['y_pred']=y_test,y_pred
df9['y_pred']=round(df9['y_preds_r']).astype(int)
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n7# df9')
print(df9.tail())
```

对应的输出信息是：

```
7# df9
   x1  x2  x3  x4  y_preds_r  y_test  y_pred
33  6.4  2.8  5.6  2.1  1.551677      1      2
34  5.8  2.8  5.1  2.4  1.209887      1      1
```

```

35  5.3  3.7  1.5  0.2  2.093058      2      2
36  5.5  2.3  4.0  1.3  2.317451      3      2
37  5.2  3.4  1.4  0.2  2.300976      2      2

```

读者可以自己打开结果文件，查看相关的结果，其中 `y_test` 是实盘真实的数据结果，`y_pred` 是程序生成的结果数据。

第 8 组代码，检验测试结果：

```

#8
dacc=zai.ai_acc_xed(df9,1,False)
print('\n8# mx:mx_sum,kok:{0:.2f}%'.format(dacc))

```

在案例中，测试结果数据使用的是 `ztop_ai` 极宽智能模块库中的 `zai.ai_acc_xed` 函数，这个函数目前有些超前，在后面的章节再进行详细讲解。

第 8 组代码对应的输出信息是：

```

8# mx:mx_sum,kok:44.74%

```

由结果数据可以看出，线性回归的准确率有些低，只有 44.74%，不过这个案例是三选一，相比 33% 的随机概率还是提高了不少。

至此，一个完整的人工智能、机器学习程序就完成了。虽然有些简陋，但毕竟是一个开始。众所周知，从 0 到 1 是一个艰难而又漫长的过程，也是一种质变的过程。

月球的一小步，人类的一大步。至此，我们迈出了人工智能、机器学习的第一步。

11

第 11 章

机器学习经典算法案例（上）

前面我们说过，Sklearn 中常用的经典机器学习算法有：线性回归、朴素贝叶斯算法、KNN 近邻算法、逻辑回归算法、随机森林算法、决策树算法、GBDT 迭代决策树算法、SVM 向量机算法和 SVM-cross 向量机交叉算法。

前面已经学习了线性回归算法，本章通过具体的案例，逐一讲解有关的人工智能、机器学习经典算法。

需要注意的是，虽然本书和有关文档经常将 Sklearn 中有关的机器学习算法称之为“xx 机器学习函数”，但其定义都是 class 类，读者要记住这一点。

11.1 线性回归

在 Sklearn 模块库中，有多种不同的线性回归函数，都位于 Linear_model 模块，函数名分别如下。

- Ridge: 岭回归函数，如图 11-1 所示。
- LASSO (Least Absolute Shrinkage and Selection Operator): 最小绝对值收缩和选择算法，俗称套索算法，如图 11-2 所示。
- MultiTaskLasso: 多任务 Lasso 套索回归算法，如图 11-3 所示。
- ElasticNet: 弹性网眼算法，如图 11-4 所示。

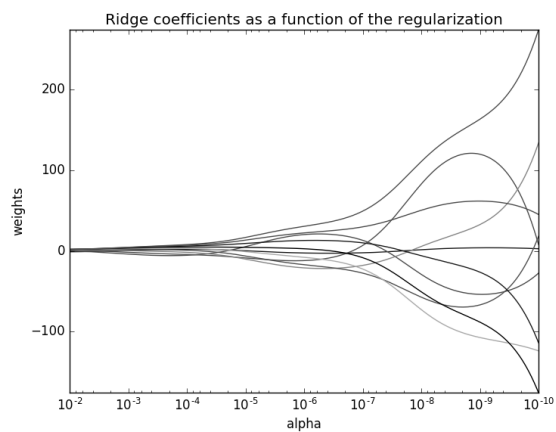


图 11-1 岭回归算法

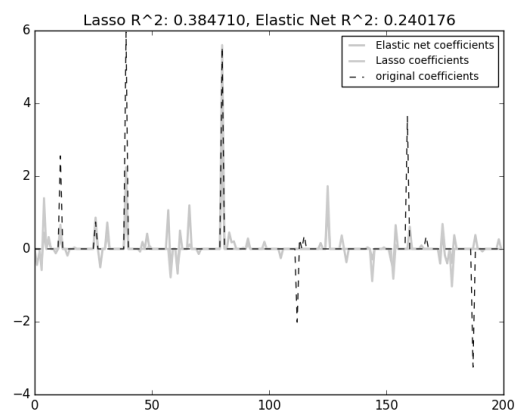


图 11-2 最小绝对值收缩和选择算法

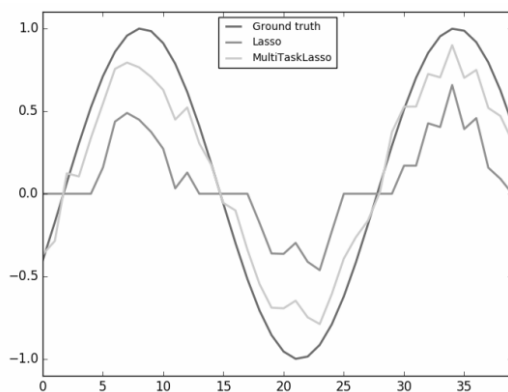


图 11-3 多任务 Lasso 套索回归算法

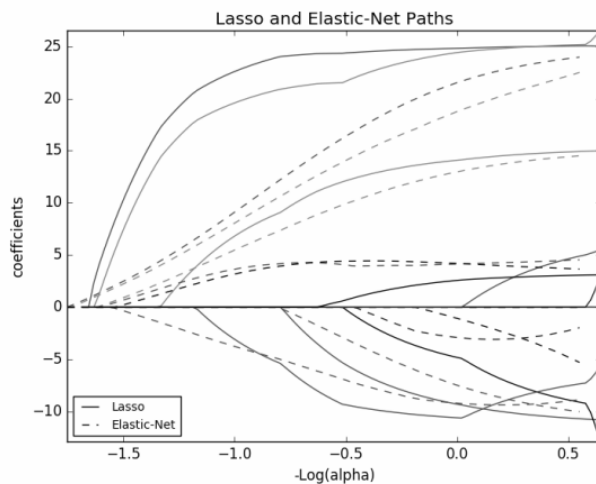


图 11-4 弹性网眼算法

- LassoLars: Lars 套索算法，如图 11-5 所示。
- OrthogonalMatchingPursuit: 正交匹配追踪（OMP）算法，如图 11-6 所示。
- BayesianRidge: 贝叶斯岭回归算法，如图 11-7 所示。

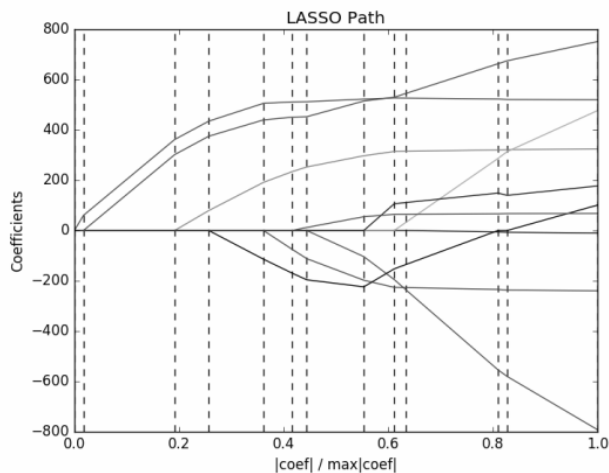


图 11-5 Lars 套索算法

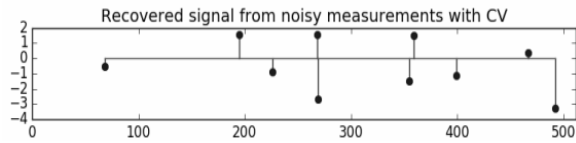


图 11-6 正交匹配追踪（OMP）算法

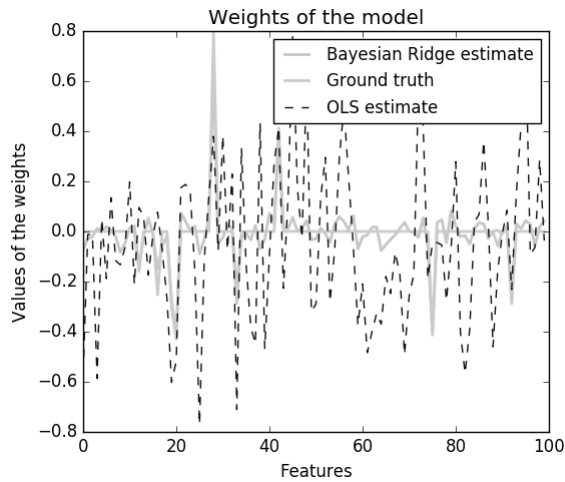


图 11-7 贝叶斯岭回归算法

- ARDRegression: ARD 自相关回归算法，如图 11-8 所示。
- LogisticRegression: 逻辑回归算法，如图 11-9 所示。

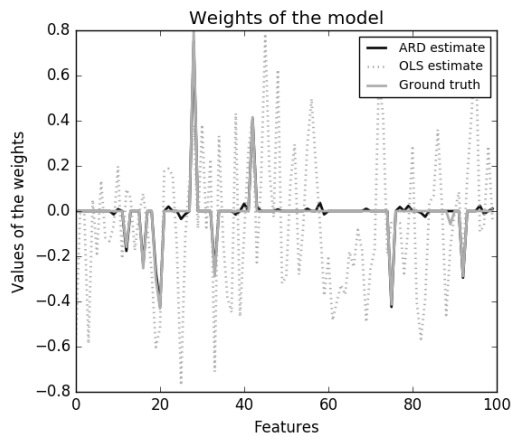


图 11-8 ARD 自相关回归算法

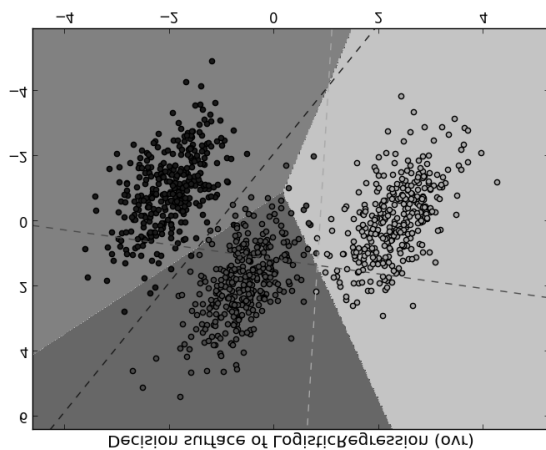


图 11-9 逻辑回归算法

- SGDClassifier: SGD 随机梯度下降算法，如图 11-10 所示。
- MultiTaskElasticNet: 多任务 ElasticNet 弹性网眼算法。
- LARS: 最小角回归算法。
- Perceptron: 感知器算法，如图 11-11 所示。

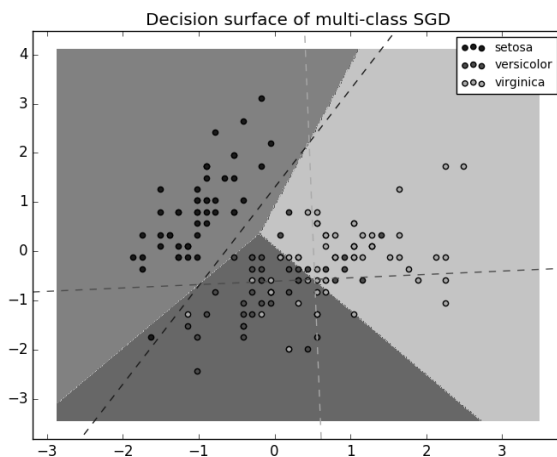


图 11-10 SGD 随机梯度下降算法

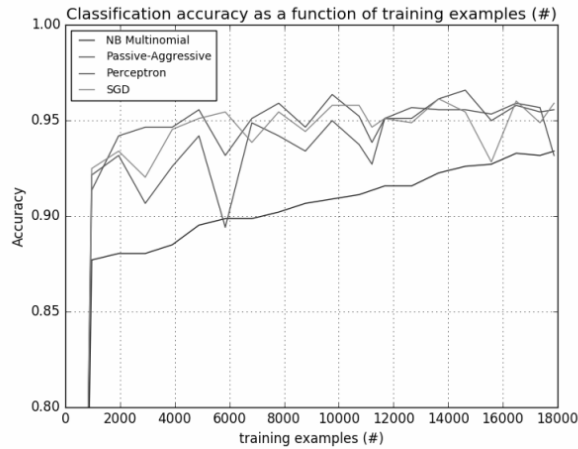


图 11-11 感知器算法

- PassiveAggressiveClassifier: PA 被动感知算法，如图 11-12 所示。
- RANSACRegressor: 鲁棒回归算法，如图 11-13 所示。
- HuberRegressor: Huber 回归算法，如图 11-14 所示。
- TheilSenRegressor: Theil-Sen 回归算法，如图 11-15 所示。
- PolynomialFeatures: 多项式函数回归算法，如图 11-16 所示。
- LinearRegression(): 最小二乘法线性回归函数。

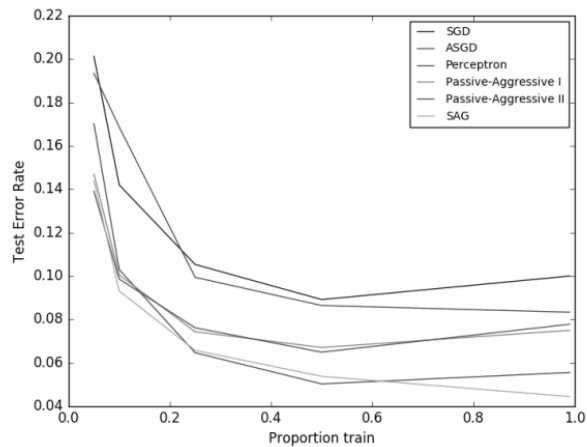


图 11-12 PA 被动感知算法

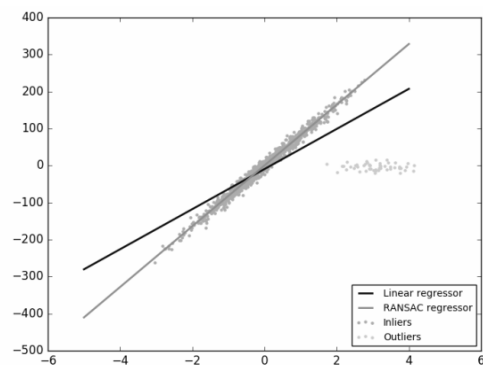


图 11-13 鲁棒回归算法

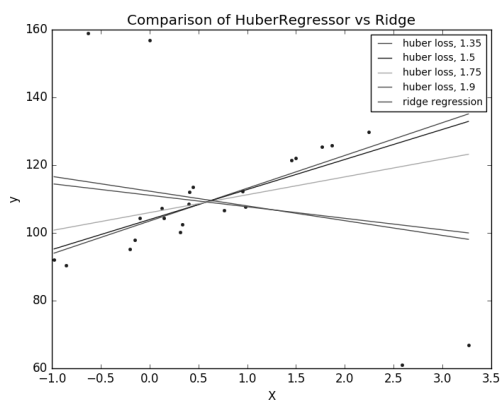


图 11-14 Huber 回归算法

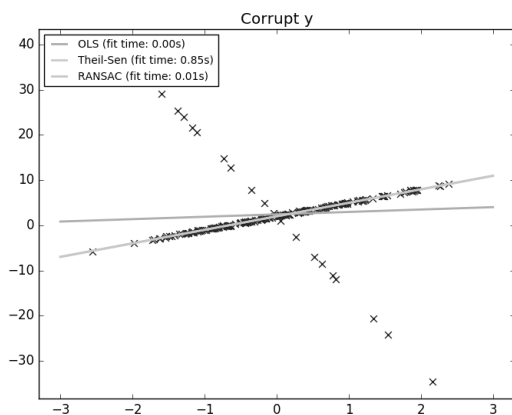


图 11-15 Theil-Sen 回归算法

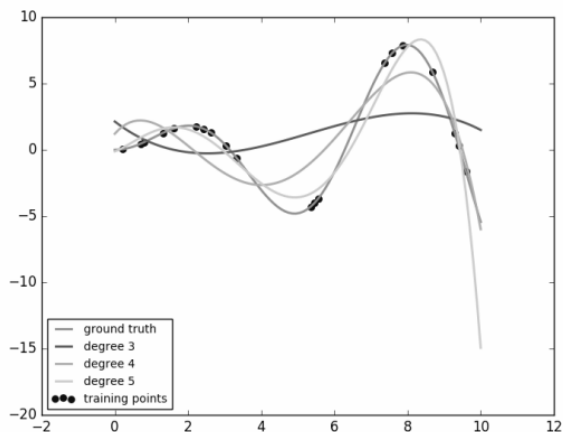


图 11-16 多项式函数回归算法

这些只是 Sklearn 0.18 版本的 `linear_model` 模块中所包括的机器学习的主要算法函数，以上函数需要注意以下事项。

- **LogisticRegression**（逻辑回归算法），虽然也属于线性回归算法，但由于用得较多，所以将其作为一个独立类别。
- **ElasticNet**（弹性网眼算法），对弹性神经网络进行分析，是一个改进型的弹性网络算法。
- **Lasso** 算法与岭回归函数和 **Lars** 算法类似。**Lasso** 算法与岭回归函数都是通过增加惩罚函数来判断、消除特征间的共线性。**Lasso** 算法与 **LARS** 算法都可以用做参数选择，得出一个相关系数的稀疏向量。

Sklearn 项目官方网站有大量的人工智能、机器学习专业文档和源码可以下载，网址是 <http://scikit-learn.org>。

11.2 逻辑回归算法

逻辑回归算法（**LogisticRegression**）虽然也属于线性回归算法，但由于使用得较多，将其作为一个独立类别，如图 11-17 所示。

Logistic 回归又称 **Logistic** 回归分析，是一种广义的线性回归分析模型，常用于数据挖掘、疾病自动诊断、经济预测等领域。例如，探讨引发疾病的危险因素，并根据危险因素预测疾病发生的概率等。

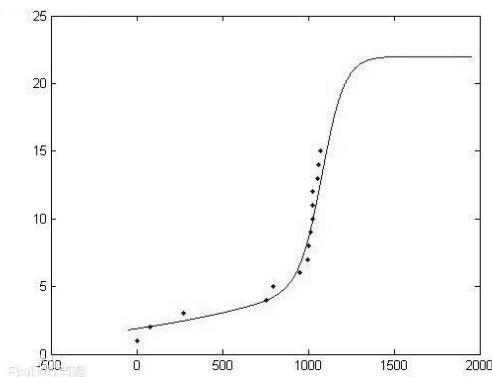


图 11-17 逻辑回归算法

逻辑回归算法函数 (LogisticRegression)，位于 Sklearn 模块库的 linear_model 线性回归子模块，函数接口是：

```
LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0,
fit_intercept=True, intercept_scaling=1, class_weight=None,
random_state=None, solver='liblinear', max_iter=100, multi_class='ovr',
verbose=0, warm_start=False, n_jobs=1)
```

11.2.1 案例 11-1：逻辑回归算法

案例 11-1 的文件名是 zai201_mx_log.py，下面具体介绍逻辑回归算法的应用。

案例 11-1 基于案例 10-4 线性回归算法，但由于案例 10-4 是第一个机器学习案例，为了方便讲解，其中有不少冗余代码，案例 11-1 进行了优化，更加接近实盘程序。

下面详细讲解案例 11-1 的程序代码。

第 1 组代码，读取训练数据并保存到相关变量，复制 x_test 测试数据到 df9 结果数据变量中：

```
#1
fs0='dat/iris_'
print('\n1# init, fs0,',fs0)
x_train=pd.read_csv(fs0+'xtrain.csv',index_col=False);
y_train=pd.read_csv(fs0+'ytrain.csv',index_col=False);
x_test=pd.read_csv(fs0+'xtest.csv',index_col=False)
y_test=pd.read_csv(fs0+'ytest.csv',index_col=False)
```

```
df9=x_test.copy()
```

第 2 组代码，根据逻辑回归算法建立机器学习的模型并保存到变量 `mx`：

```
#2
print('\n2# 建模')
mx =zai.mx_log(x_train.values,y_train.values)
```

在本案例中，第 2 组代码使用的是逻辑回归算法建立机器学习模型，但是并没有直接调用 Sklearn 模块库中 `LogisticRegression` 线性回归建模函数，而是通过 `ztop_ai` 极宽智能模块库的函数接口间接进行调用，对应的函数代码是：

```
# 逻辑回归算法，函数名，LogisticRegression
def mx_log(train_x, train_y):

    mx = LogisticRegression(penalty='l2')
    mx.fit(train_x, train_y)
    return mx
```

`LogisticRegression` 回归函数，位于 `sklearn.linear_model` 模块，函数接口在前面已经介绍过了，在此不再赘述。

第 3 组代码，运行机器学习变量 `mx` 的内置函数 `predict`，生成结果数据，并保存到结果数据变量 `df9`：

```
#3
print('\n3# 预测')
y_pred = mx.predict(x_test.values)
df9['y_preds_r']=y_pred
df9['y_test'],df9['y_pred']=y_test,y_pred
df9['y_pred']=round(df9['y_preds_r']).astype(int)
```

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

对应的输出信息是：

```
4# df9
   x1  x2  x3  x4  y_preds_r  y_test  y_pred
33  6.4  2.8  5.6  2.1         1         1         1
34  5.8  2.8  5.1  2.4         1         1         1
35  5.3  3.7  1.5  0.2         2         2         2
```

```
36 5.5 2.3 4.0 1.3      3      3      3
37 5.2 3.4 1.4 0.2      2      2      2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

对应的输出信息是：

```
5# mx:mx_sum,kok:84.21%
```

84.21%的结果非常不错了，案例 10-4 线性回归的准确率只有 44.74%：

```
8# mx:mx_sum,kok:44.74%
```

案例 11-1 虽然将程序进行了优化，但还是有不少冗余代码，在实盘操作时只要前面三组代码就足够了，而且有些命令还可以进一步精简。

11.3 朴素贝叶斯算法

学过统计学的读者一定知道贝叶斯定理，这个 250 多年前发明的算法，在信息领域内有着无与伦比的地位。贝叶斯分类是一系列分类算法的总称，这类算法均以贝叶斯定理为基础，故统称为贝叶斯分类。

朴素贝叶斯算法（Naive Bayesian）是其中应用最为广泛的分类算法之一。朴素贝叶斯分类器基于一个简单的假定：给定目标值时，属性之间相互条件独立，如图 11-18 所示。

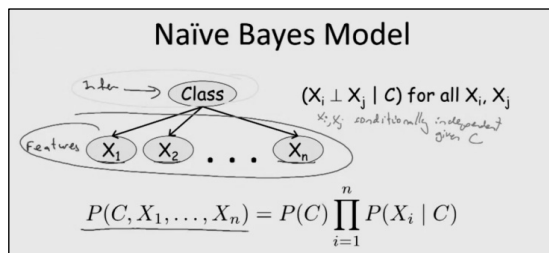


图 11-18 朴素贝叶斯算法

在 Sklearn 模块库中，涉及贝叶斯原理的算法函数有很多，这里只介绍位于独立子模块 naive_bayes 中的几个相关函数。

- MultinomialNB，多项式朴素贝叶斯算法，如图 11-19 所示。

- GaussianNB，高斯朴素贝叶斯算法，如图 11-20 所示。
- BernoulliNB，伯努利朴素贝叶斯算法。

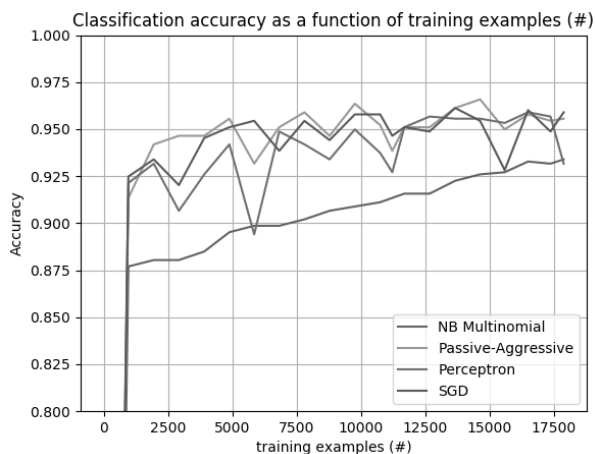


图 11-19 多项式朴素贝叶斯算法

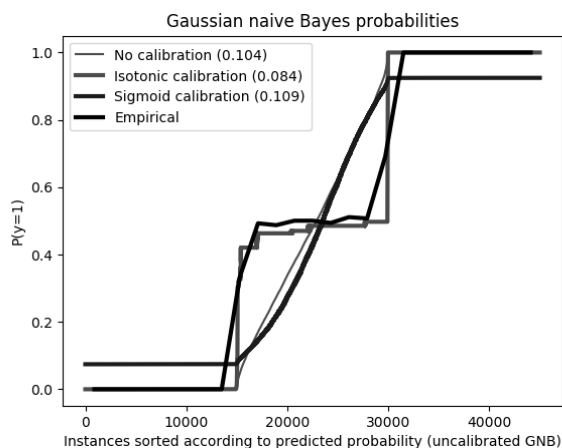


图 11-20 高斯朴素贝叶斯算法

11.3.1 案例 11-2：贝叶斯算法

案例 11-2 的文件名是 `zai202_mx_nb.py`，介绍的是多项式朴素贝叶斯算法，函数名 `MultinomialNB` 位于 `naive_bayes` 模块，函数接口是：

```
MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)
```

案例 11-2 源码和本章其他小节的源码，基本都与案例 11-1 的源码类似，只是调用的具体的机器学习函数不同，为简化篇幅，本节及后面的章节只介绍不同的程序代码部分。

案例 11-2 的核心在第 2 组的建模代码：

```
#2
print('\n2# 建模')
mx =zai.mx_bayes(x_train.values,y_train.values)
```

本案例中的第 2 组代码使用的是多项式朴素贝叶斯算法，建立机器学习模型，但是并没有直接调用 Sklearn 模块库中 MultinomialNB 建模函数，而是通过 ztop_ai 极宽智能模块库的函数接口间接进行调用，对应的函数代码是：

```
# 多项式朴素贝叶斯算法, Multinomial Naive Bayes, 函数名, multinomialnb
def mx_bayes(train_x, train_y):
    mx = MultinomialNB(alpha=0.01)
    mx.fit(train_x, train_y)
    return mx
```

MultinomialNB 回归函数位于 sklearn.naive_bayes 模块，函数接口我们在前面已经介绍过了，在此不再赘述。

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

对应的输出信息是：

```
4# df9
      x1  x2  x3  x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1         1         1         1
34  5.8  2.8  5.1  2.4         1         1         1
35  5.3  3.7  1.5  0.2         2         2         2
36  5.5  2.3  4.0  1.3         1         3         1
37  5.2  3.4  1.4  0.2         2         2         2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```


对应的输出信息是：

```
5# mx:mx_sum,kok:57.89%
```

57.89%的结果有些偏低，虽然比线性回归算法的 44.74%的结果好一点，但远远低于逻辑回归算法的 84.21%。

11.4 KNN近邻算法

KNN 近邻算法，又叫做 K 最近邻（KNN，K-NearestNeighbor）分类算法，是数据挖掘分类技术中最简单的方法之一。所谓 K 最近邻就是 K 个最近的邻居的意思，即每个样本都可以用它最接近的 K 个邻居来代表，如图 11-21 所示。

在 Sklearn 模块库中，KNN 近邻算法相关的算法函数位于 Neighbors 模块库，主要的机器学习算法函数如下。

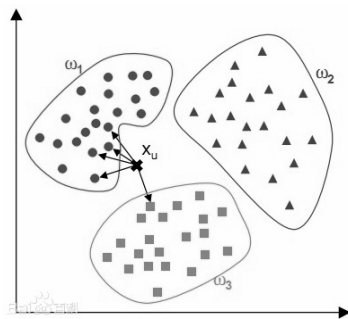


图 11-21 K 近邻示意图

- KNeighborsClassifier: KNN 近邻算法，如图 11-22 所示。

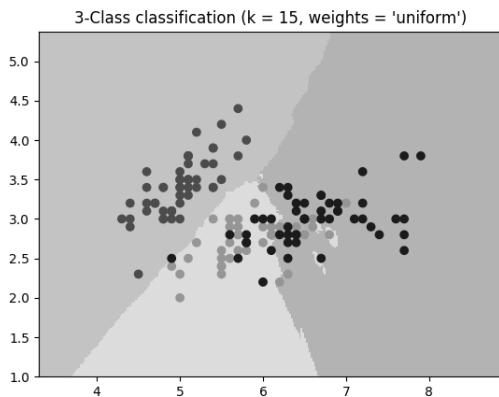


图 11-22 KNN 近邻算法

- NearestNeighbors: 最近邻居算法，如图 11-23 所示。
- KNeighborsRegressor: K 近邻回归算法，如图 11-24 所示。
- NearestCentroid: 最近质心算法，如图 11-25 所示。

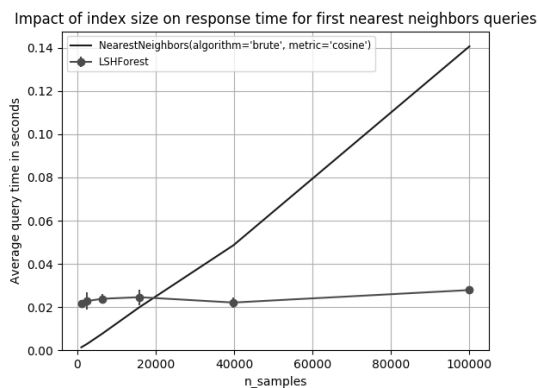


图 11-23 最近邻居算法

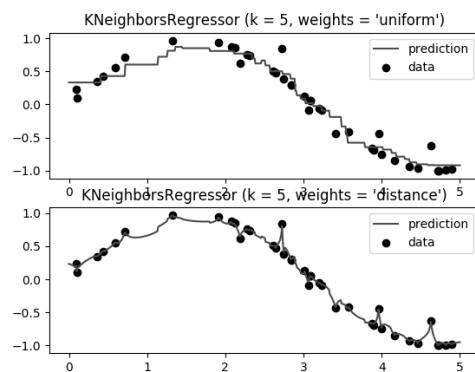


图 11-24 K 近邻回归算法

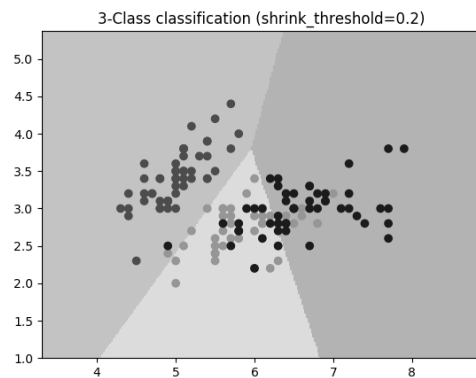


图 11-25 最近质心算法

- LSHForest (Locality Sensitive Hashing forest, LSH): 局部敏感哈希森林算法, 是最近邻搜索方法的代替, 排序实现二进制搜索和 32 位定长数组和散列, 使用 hash 家族的随机投影方法, 近似余弦距离, 如图 11-26 所示。

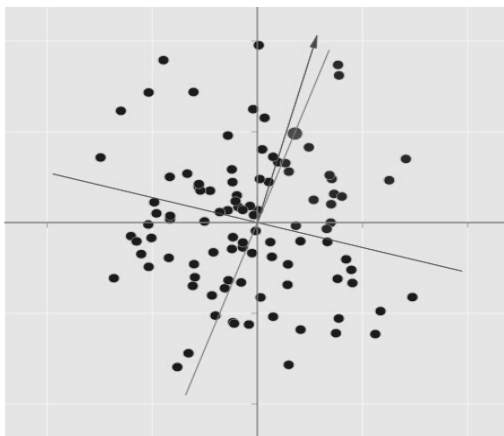


图 11-26 局部敏感哈希森林算法

11.4.1 案例 11-3: KNN 近邻算法

案例 11-3 的文件名是 zai203_mx_knn.py, 介绍 KNN 近邻算法, 位于 Neighbors 模块, 函数名是 KNeighborsClassifier, 函数接口是:

```
KNeighborsClassifier(n_neighbors=5, weights='uniform',
algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None,
n_jobs=1, **kwargs)
```

案例 11-3 的核心是第 2 组的建模代码:

```
#2
print('\n2# 建模')
mx =zai.mx_knn(x_train.values,y_train.values)
```

在本案例中, 第 2 组代码使用的是 KNN 近邻算法, 建立机器学习模型, 但是并没有直接调用 Sklearn 模块库中的 KNeighborsClassifier 函数, 而是通过 ztop_ai 极宽智能模块库的函数接口间接进行调用, 对应的函数代码是:

```
# KNN 近邻算法, 函数名, KNeighborsClassifier
def mx_knn(train_x, train_y):
```

```
mx = KNeighborsClassifier()
mx.fit(train_x, train_y)
return mx
```

KNN 近邻算法，位于 `sklearn.neighbors` 模块，函数接口我们在前面已经介绍过了，在此不再赘述。

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

对应的输出信息是：

```
4# df9
      x1  x2  x3  x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1         1         1         1
34  5.8  2.8  5.1  2.4         1         1         1
35  5.3  3.7  1.5  0.2         2         2         2
36  5.5  2.3  4.0  1.3         3         3         3
37  5.2  3.4  1.4  0.2         2         2         2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

对应的输出信息是：

```
5# mx:mx_sum,kok:100.00%
```

这个结果居然是 100%，不过读者不要高兴得太早，这个 100%只是基于爱丽丝数据集的，数据量有些偏少，不一定有普适性。

不管如何，这个案例也是读者学习人工智能、机器学习算法的第一个 100%准确率的程序，第一个全垒打程序，还是值得庆祝一下的。

11.5 随机森林算法

随机森林（Random Forest）算法是指利用多棵树对样本进行训练并预测的一种算法，如图 11-27 所示。随机森林这个术语是由 1995 年贝尔实验室的 Tin Kam Ho

所提出的随机决策森林(Random Decision Forests)发展而来的。Leo Breiman 和 Adele Cutler 推论出了随机森林的算法，并注册了 Random Forests 商标。

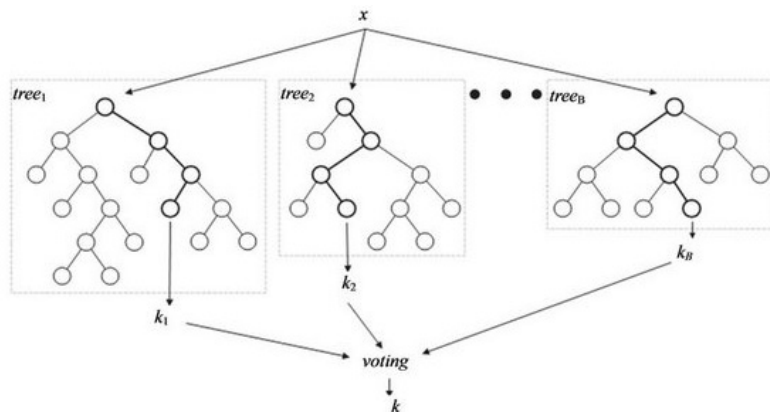


图 11-27 随机森林示意图

随机森林算法是一个包含多个决策树的算法，并且其输出的类别是由个别树输出的类别的众数而定，这个算法结合了 Breimans 的“Bootstrap Aggregating”想法和 Ho 的“Random Subspace Method”算法，以建造决策树的集合。

在 Sklearn 模块库中，随机森林算法相关的算法函数，位于集成算法模块 ensemble 中，其中相关的机器学习算法函数如下。

- RandomForestClassifier: 随机森林算法，如图 11-28 所示。

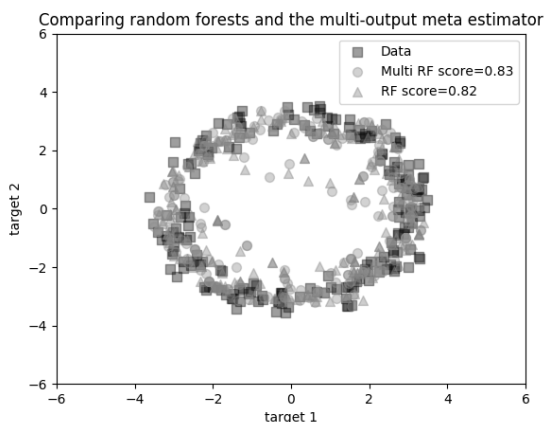


图 11-28 随机森林算法

- **BaggingClassifier**: 装袋算法, 相当于多个专家投票表决, 对于多次测试, 每个样本返回的是多次预测结果较多的那个, 如图 11-29 所示。

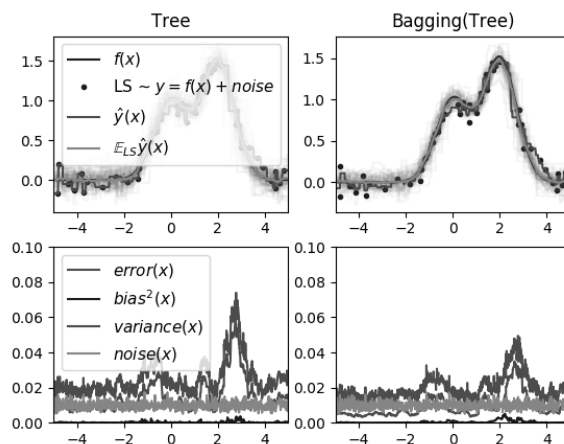


图 11-29 装袋算法

- **ExtraTreesClassifier**: 完全随机树算法, 如图 11-30 所示。

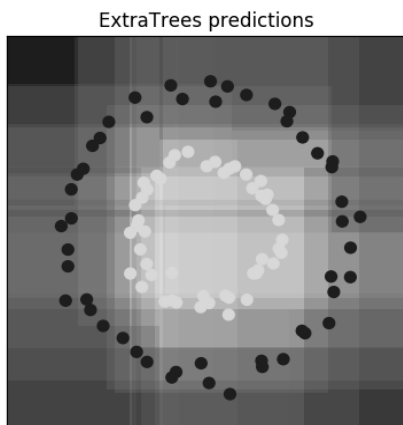


图 11-30 完全随机树算法

- **Adaboost**: 迭代算法, 其核心思想是针对同一个训练集训练不同的分类器 (弱分类器), 然后把这些弱分类器集合起来, 构成一个更强的最终分类器 (强分类器), 如图 11-31 所示。

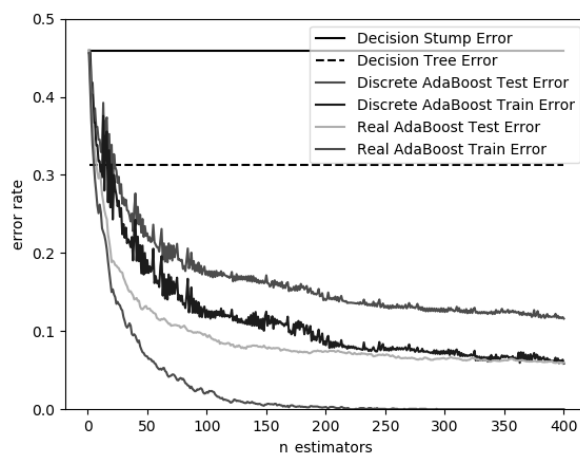


图 11-31 迭代算法

- GradientBoostingClassifier: GBT 梯度 Boosting 树算法, 如图 11-32 所示。
- GradientBoostingRegressor: 梯度回归算法, 如图 11-33 所示。
- VotingClassifier: 投票算法, 如图 11-34 所示。

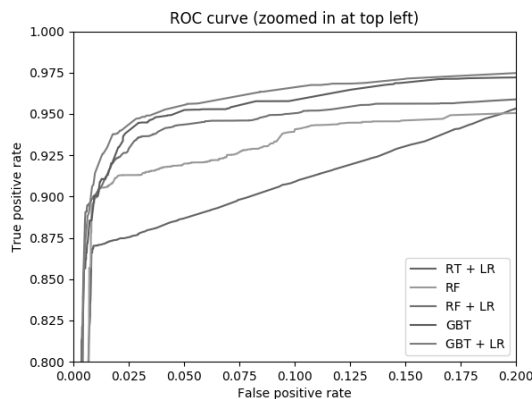


图 11-32 GBT 梯度 Boosting 树算法

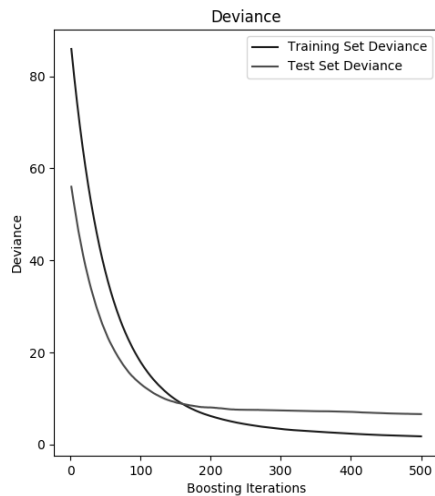


图 11-33 梯度回归算法

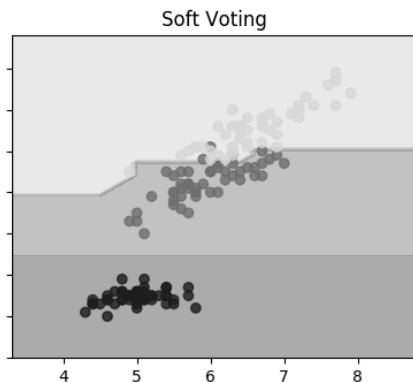


图 11-34 投票算法

11.5.1 案例 11-4：随机森林算法

案例 11-4 的文件名是 zai204_mx_rf.py，介绍随机森林算法，位于 Ensemble 集成算法模块，函数名是 KNeighborsClassifier，函数接口是：

```
RandomForestClassifier(n_estimators=10, criterion='gini', max_
depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_
fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_split=1e-07, bootstrap=True, oob_score=False, n_jobs=1,
random_state=None, verbose=0, warm_start=False, class_weight=None)
```

案例 11-4 的核心是第 2 组的建模代码：

```
#2
print('\n2# 建模')
mx =zai.mx_forest (x_train.values,y_train.values)
```

在本案例中，第 2 组代码使用随机森林算法建立机器学习模型，但是并没有直接调用 Sklearn 模块库中的 RandomForestClassifier 函数，而是通过 ztop_ai 极宽智能模块库的函数接口间接进行调用，对应的函数代码是：

```
# 随机森林算法, Random Forest Classifier, 函数名, RandomForestClassifier
def mx_forest(train_x, train_y):
    mx = RandomForestClassifier(n_estimators=8)
    mx.fit(train_x, train_y)
    return mx
```


随机森林算法位于 `sklearn.ensemble` 集成算法模块，函数接口我们在前面已经介绍过了，在此不再赘述。

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

对应的输出信息是：

```
4# df9
      x1  x2  x3  x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1         1        1        1
34  5.8  2.8  5.1  2.4         1        1        1
35  5.3  3.7  1.5  0.2         2        2        2
36  5.5  2.3  4.0  1.3         3        3        3
37  5.2  3.4  1.4  0.2         2        2        2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

对应的输出信息是：

```
5# mx:mx_sum,kok:97.37%
```

结果是 97.37%，已经非常好了，虽然不是 100% 的全垒打，不过作为预测模型，这种精度比 100% 给人的感觉更加心安。

12

第 12 章

机器学习经典算法案例（下）

12.1 决策树算法

决策树算法最早产生于 20 世纪 60 年代，本质上决策树是通过一系列规则对数据进行分类的过程。

20 世纪 70 年代末，J.Ross Quinlan 提出了 ID3 算法，此算法的目的在于减少树的深度，但是忽略了叶子数目的研究。C4.5 算法在 ID3 算法的基础上进行了改进，在预测变量的缺值处理、剪枝技术、派生规则等方面做了较大改进，既适合于分类问题，又适合于回归问题。

决策树算法是一种逼近离散函数值的方法，也是一种典型的分类方法，其首先对数据进行处理，利用归纳算法生成可读的规则和决策树，然后使用决策对新数据进行分析。决策树算法构造决策树来发现数据中蕴涵的分类规则。构造精度高、规模小的决策树是决策树算法的核心内容，如图 12-1 所示。

在 Sklearn 模块库中，与 DecisionTreeClassifier 决策树算法相关的算法函数位于 Tree 模块中，其中相关的机器学习算法函数如下。

- DecisionTreeClassifier: 决策树算法，如图 12-2 所示。
- DecisionTreeRegressor: 决策树回归算法，如图 12-3 所示。
- ExtraTreeClassifier: 完全随机树算法。



图 12-1 决策树算法的核心内容

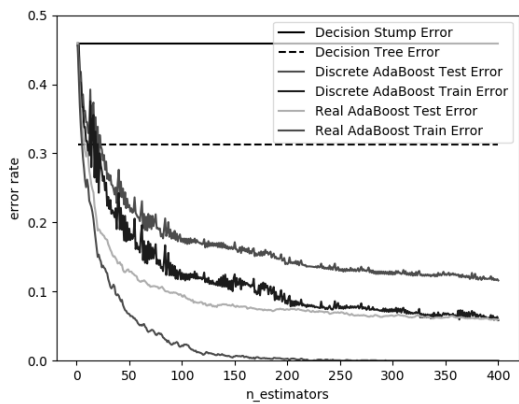


图 12-2 决策树算法

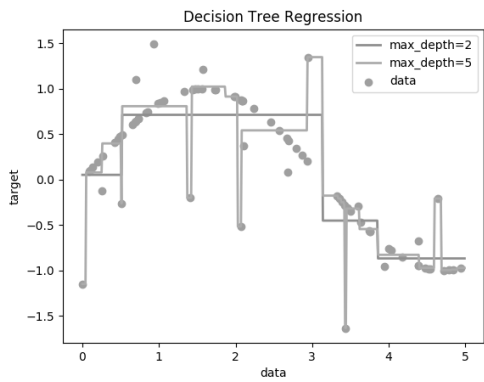


图 12-3 决策树回归算法

- ExtraTreeRegressor: 完全随机树回归算法。
- export_graphviz: 辅助函数, 输出决策树图形, 如图 12-4 所示。

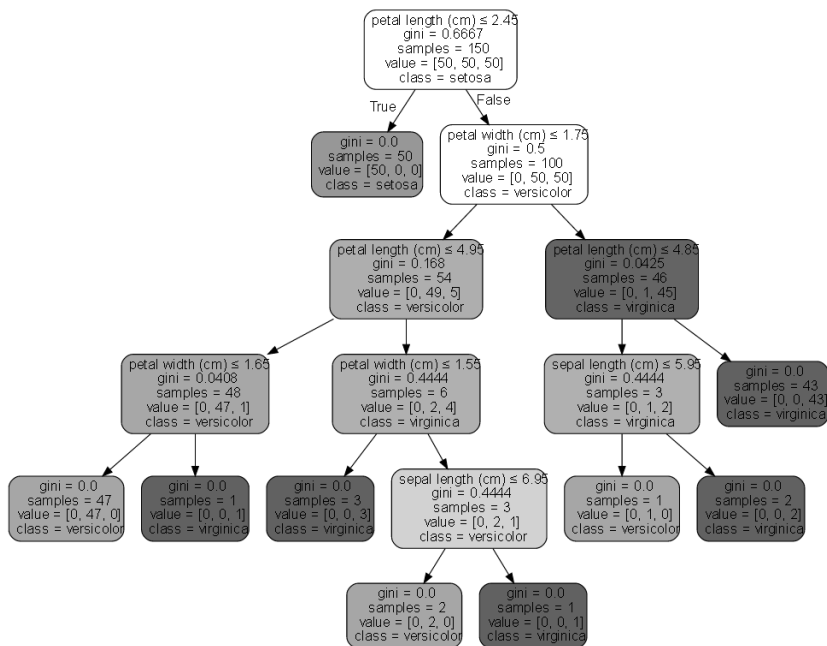


图 12-4 决策树图形

12.1.1 案例 12-1: 决策树算法

案例 12-1 的文件名是 zai301_mx_dtree.py, 介绍决策树算法, 函数位于 Tree 决策树模块, 函数名是 DecisionTreeClassifier, 函数接口是:

```
DecisionTreeClassifier(criterion='gini', splitter='best', max_
depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_
fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_
nodes=None, min_impurity_split=1e-07, class_weight=None, presort= False)
```

案例 12-1 的核心是第 2 组的建模代码:

```
#2
print('\n2# 建模')
mx =zai.mx_dtree (x_train.values,y_train.values)
```

在本案例中，第 2 组代码使用 Decision Tree 决策树算法建立机器学习模型，但是并没有直接调用 Klearn 模块库中的 DecisionTreeClassifier 函数，而是通过 ztop_ai 极宽智能模块库的函数接口间接进行调用，对应的函数代码是：

```
# 决策树算法，函数名，tree.DecisionTreeClassifier()
def mx_dtree(train_x, train_y):
    mx = tree.DecisionTreeClassifier()
    mx.fit(train_x, train_y)
    return mx
```

Decision Tree 决策树算法位于 sklearn.tree 模块，函数接口在前面已经介绍过了，在此不再赘述。

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

对应的输出信息是：

```
4# df9
      x1  x2  x3  x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1         1        1        1
34  5.8  2.8  5.1  2.4         1        1        1
35  5.3  3.7  1.5  0.2         2        2        2
36  5.5  2.3  4.0  1.3         3        3        3
37  5.2  3.4  1.4  0.2         2        2        2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

对应的输出信息是：

```
5# mx:mx_sum,kok:97.37%
```

Decision Tree 决策树算法的测试结果也是 97.37%，与 KNN 近邻算法的结果一样。

12.2 GBDT 迭代决策树算法

目前 GBDT 迭代决策树算法非常火热，因为它在近期多次国际级别的人工智能算法大赛中，如 Kaggle，都获得了胜利。

GBDT 全称为 Gradient Boosting Decision Tree，是一种基于决策树（Decision Tree）实现的分类回归算法。

GBDT 算法由多棵决策树组成，所有树的结论累加起来作为最终结果。它在被提出之初就和 SVM 一起被认为是泛化能力（Generalization）较强的算法。近年来更因为被用于搜索排序的机器学习模型而引起关注，如图 12-5 所示。

GBDT 是一个应用很广泛的算法，可以用来做分类和回归，在很多的數據上都有不错的效果。GBDT 算法还有一些其他的名字，比如 MART（Multiple Additive Regression Tree）、GBRT（Gradient Boost Regression Tree）、Tree Net 等。



图 12-5 GBDT 算法决策树

12.2.1 案例 12-2：GBDT 迭代决策树算法

案例 12-2 的文件名是 zai302_mx_gbd.py，介绍 GBDT 迭代决策树算法，函数位于 Ensemble 集成算法模块，函数名是 GradientBoostingClassifier，函数接口是：

```
GradientBoostingClassifier(loss='deviance', learning_rate=0.1,
n_estimators=100, subsample=1.0, criterion='friedman_mse', min_
samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_depth=3, min_impurity_split=1e-07, init=None, random_state= None,
max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False,
presort='auto')
```

案例 12-2 的核心是第 2 组的建模代码：

```
#2
print('\n2# 建模')
mx =zai.mx_GBDT (x_train.values,y_train.values)
```

在本案例中，第 2 组代码使用 GBDT 迭代决策树算法建立机器学习模型，但是并没有直接调用 Sklearn 模块库中的 GradientBoostingClassifier 函数，而是通过

ztop_ai 极宽智能模块库的函数接口间接进行调用，对应的函数代码是：

```
# GBDT 迭代决策树算法, Gradient Boosting Decision Tree,
# 又叫 MART(Multiple Additive Regression Tree)
def mx_GBDT(train_x, train_y):
    mx = GradientBoostingClassifier(n_estimators=200)
    mx.fit(train_x, train_y)
    return mx
```

GBDT 迭代决策树算法位于 sklearn.ensemble 集成算法模块，函数接口我们在前面已经介绍过了，在此不再赘述。

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

对应的输出信息是：

```
4# df9
      x1  x2  x3  x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1         1         1         1
34  5.8  2.8  5.1  2.4         1         1         1
35  5.3  3.7  1.5  0.2         2         2         2
36  5.5  2.3  4.0  1.3         3         3         3
37  5.2  3.4  1.4  0.2         2         2         2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

对应的输出信息是：

```
5# mx:mx_sum,kok:97.37%
```

无巧不成书，GBDT 迭代决策树算法的测试结果居然也是 97.37%，与 Decision Tree 决策树算法、KNN 近邻算法一样。

12.3 SVM 向量机

SVM 全称是 Support Vector Machine，即支持向量机，是一种有监督的机器学习算法，通常用来进行模式识别、分类及回归分析，简称 SVM 向量机，如图 12-6 所示。

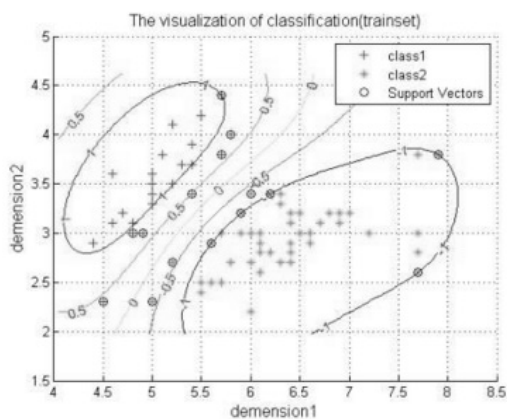


图 12-6 SVM 向量机

在 Sklearn 模块库中，有独立的 SVM 向量机模块 `sklearn.svm`，其中相关的机器学习算法函数如下。

- SVC: 支持向量机算法。
- LinearSVC: 线性向量算法。
- NuSVC: Nu 支持向量算法。
- SVR: SVR (TEpsilon) 支持向量算法。
- NuSVR: Nu 支持 SVR 向量算法。
- OneClassSVM: 一类支持向量机异常的检测算法。
- `ll_min_c`: 辅助函数，返回边界参数。

有关函数效果如图 12-7 所示。

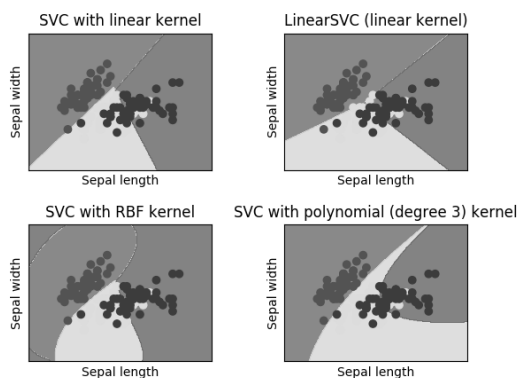


图 12-7 SVM 向量机算法

12.3.1 案例 12-3: SVM 向量机算法

案例 12-3 的文件名是 `zai303_mx_svm.py`，介绍 SVM 向量机算法，位于 SVM 算法模块，函数名是 `KNeighborsClassifier`，函数接口是：

```
SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0,
shrinking=True, probability=False, tol=0.001, cache_size=200,
class_weight=None, verbose=False, max_iter=-1, decision_function_
shape=None, random_state=None)
```

案例 12-3 的核心是第 2 组的建模代码：

```
#2
print('\n2# 建模')
mx =zai.mx_svm (x_train.values,y_train.values)
```

在本案例中，第 2 组代码使用 SVM 向量机算法建立机器学习模型，但是并没有直接调用 Sklearn 模块库中的 SVC 向量机函数，而是通过 `ztop_ai` 极宽智能模块库的函数接口间接进行调用，对应的函数代码是：

```
# SVM 向量机算法，函数名，SVC
def mx_svm(train_x, train_y):
    mx = SVC(kernel='rbf', probability=True)
    mx.fit(train_x, train_y)
    return mx
```

SVM 向量机算法位于 `sklearn.svm` 模块，函数接口在前面已经介绍过了，在此不再赘述。

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

对应的输出信息是：

```
4# df9
      x1  x2  x3  x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1         1         1         1
34  5.8  2.8  5.1  2.4         1         1         1
35  5.3  3.7  1.5  0.2         2         2         2
36  5.5  2.3  4.0  1.3         3         3         3
37  5.2  3.4  1.4  0.2         2         2         2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

对应的输出信息是：

```
5# mx:mx_sum,kok:97.37%
```

SVM 向量机算法的结果也是 97.37%。

12.4 SVM-cross 向量机交叉算法

SVM-cross 向量机交叉算法其实也是一种 SVM 向量机算法，函数定义如下：

```
# SVM-cross 向量机交叉算法，函数名，SVC
def mx_svm_cross(train_x, train_y):

    mx = SVC(kernel='rbf', probability=True)
    param_grid={'C': [1e-3, 1e-2, 1e-1, 1, 10, 100, 1000], 'gamma': [0.001,
0.0001]}
    grid_search = GridSearchCV(mx, param_grid, n_jobs = 1, verbose=1)
    grid_search.fit(train_x, train_y)
    best_parameters = grid_search.best_estimator_.get_params()
    #for para, val in best_parameters.items():
    #    print( para, val)
    mx = SVC(kernel='rbf', C=best_parameters['C'], gamma=best_
parameters['gamma'], probability=True)
    mx.fit(train_x, train_y)
    return mx
```

SVM-cross 向量机交叉算法的函数代码有些复杂，但调用接口都是一样的。

简单来说，SVM-cross 向量机交叉算法采用了交叉验证（Cross Validation，CV）模式，在第一次调用 SVC 函数时，从结果中选出最好的参数，再次调用 SVC 函数时，采用类似迭代的方式。

交叉验证是用来验证机器学习性能的一种统计分析方法，基本思想是在某种环境下将原始数据（Dataset）进行分组，一部分作为训练集（Train Set），另一部分作为验证集（Validation Set），首先用训练集对分类器进行训练，再利用验证集来测试训练得到的模型（Model），以此来作为评价分类器的性能指标。

12.4.1 案例 12-4: SVM-cross 向量机交叉算法

案例 12-4 的文件名是 zai304_mx_rf.py, 介绍 SVM-cross 向量机交叉算法。

SVM-cross 向量机交叉算法案例, 在某些平台可能会出现运行错误, 这个是正常现象, 读者知道该函数的调用模式即可。案例 12-4 的核心是第 2 组的建模代码:

```
#2
print('\n2# 建模')

mx =zai.mx_svm_cross (x_train.values,y_train.values)
```

在本案例中, 第 2 组代码使用 SVM-cross 向量机交叉算法建立机器学习模型, 但是并没有直接调用 Sklearn 模块库中的 SVC 函数, 而是通过 ztop_ai 极宽智能模块库的函数接口间接进行调用。

第 4 组代码, 保存数据结果并显示相关信息:

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

对应的输出信息是:

```
4# df9
      x1  x2  x3  x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1         1        1        1
34  5.8  2.8  5.1  2.4         1        1        1
35  5.3  3.7  1.5  0.2         2        2        2
36  5.5  2.3  4.0  1.3         3        3        3
37  5.2  3.4  1.4  0.2         2        2        2
```

第 5 组代码, 检验测试结果:

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

对应的输出信息是:

```
5# mx:mx_sum,kok:94.74%
```

理论上, SVM-cross 向量机交叉算法的效果应该比 SVM 向量机算法更好, 因为毕竟多一道交叉验证迭代运算。

不过案例 12-4 的准确性测试只有 94.74%, 低于 SVM 向量机算法的 97.37%, 这也是正常的。因为有时候会出现过度迭代, 这也从计算流程的角度间接证明了笔者

的小数据理论：

人工智能、机器学习不是数据量越大越好，也不是迭代计算的次数越多越好。

另外，也因为爱丽丝数据集总量太少，才 150 组数据。

12.5 神经网络算法

神经网络算法又被称为神经网络系统（Artificial Neural Networks，简称 ANN），是 20 世纪 40 年代后出现的。它是由众多的神经元可调的连接权值连接而成，具有大规模并行处理、分布式信息存储、良好的自组织和自学习能力等特点，如图 12-8 所示。

在神经网络算法中，比较经典的是 BP（Back Propagation）算法，又称为误差反向传播算法，是人工神经网络中的一种监督式的学习算法。BP 神经网络算法在理论上可以逼近任意函数，基本的结构由非线性变化单元组成，具有很强的非线性映射能力。而且网络的中间层数、各层的处理单元数及网络的学习系数等参数可以根据具体情况设定，灵活性很大，在优化、信号处理与模式识别、智能控制、故障诊断等许多领域都有着广泛的应用前景。

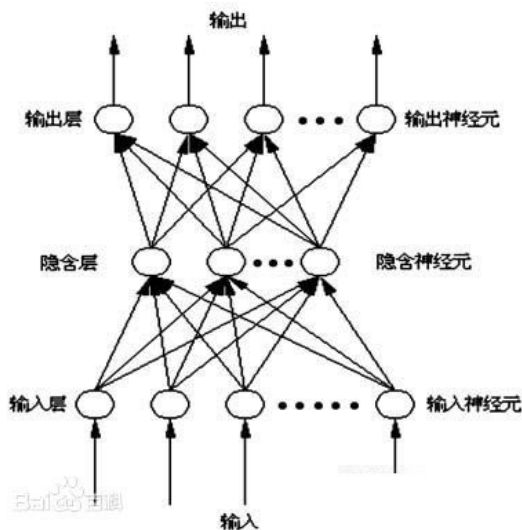


图 12-8 神经网络算法

12.5.1 经典神经网络算法

常用的神经网络算法如下。

- BP 神经网络算法，又称为误差反向传播算法，是人工神经网络中的一种监督式的学习算法。
- RBF（径向基）神经网络算法：20 世纪 80 年代末，由 J.Moody 和 C.Darken 提出，是基于径向基函数（RBF-Radial Basis Function）的神经网络算法，RBF 网络是一种局部逼近网络，能够以任意精度逼近任意连续函数，特别适合于解决分类问题。
- 感知器神经网络算法：是一个具有单层计算神经元的神经网络，网络的传递函数是线性阈值单元。
- 线性神经网络算法：是一种比较简单的神经网络算法，由一个或者多个线性神经元构成。采用线性函数作为传递函数，所以输出可以是任意值。
- 自组织神经网络算法：芬兰学者 Kohonen 根据生物神经细胞中存在的自学习特征，提出了自组织特征映射神经网络模型。他认为一个神经网络在接受外界输入模式时，会自适应地对输入信号的特征进行学习，进而自组织成不同的区域，并且在各个区域对输入模式有不同的响应特征，包括自组织竞争网络算法、自组织特征映射网络算法、学习向量量化网络算法等。
- 反馈神经网络算法：该理论认为，信息在前向传递的同时还要进行反向传递，这种信息的反馈可以发生在不同网络层的神经元之间，也可以只局限于某一层神经元上。反馈网络的典型代表是 Elman 网络算法和 Hopfield 网络算法。

目前最热门的是深度学习算法，AlphaGo、TensorFlow 主打的都是深度学习算法。

以下是百度百科关于深度学习的词条：

深度学习的概念源于人工神经网络的研究。包含多隐层的多层感知器就是一种深度学习结构。深度学习通过组合低层特征形成更加抽象的高层，表示属性类别或特征，以发现数据的分布式特征表示。

深度学习的概念由 Hinton 等人于 2006 年提出。基于深度置信网络（DBN）提出非监督贪心逐层训练算法，为解决深层结构相关的优化难题带来希望，随后提出多层自动编码器深层结构。此外，Lecun 等人提出的卷积神经网络是第一个真正的多层结构学习算法，它利用空间的相对关系减少参数数目，以提高训练性能。

深度学习是机器学习研究中的一个新领域，其动机在于建立模拟人脑进行分析学习的神经网络，模仿人脑的机制来解释数据，例如图像、声音和文本。

同机器学习方法一样，深度机器学习方法也有监督学习与无监督学习之分。不同的学习框架下建立的学习模型也有很大不同。例如，卷积神经网络（Convolutional Neural Networks, CNNs）就是一种深度的监督学习下的机器学习模型，而深度置信网（Deep Belief Nets, DBNs）就是一种无监督学习下的机器学习模型。

12.5.2 Sklearn 神经网络算法

在 Sklearn 0.17 版本以前，Sklearn 对神经网络算法的支持很弱，通常都是使用其兼容模块库 SKNN 代替，或者使用其他的 Python 模块库，比如 Theano、Pybrain、FFNN、Pylearn 2 等。

不过随着神经网络算法在人工智能、机器学习领域的异军突起，特别是 AlphaGo、TensorFlow、Torch 主打的都是深度学习算法，Sklearn 自 0.18 版本开始，强化了神经网络的功能，目前 `sklearn.neural_network` 网络模块提供了以下 3 种算法函数。

- BernoulliRBM：伯努利受限玻尔兹曼机神经网络算法，简称 RBM 算法。
- MLPClassifier：多层感知器神经网络算法，简称 MLP 算法。
- MLPRegressor：多层感知器神经网络回归算法。

12.5.3 人工智能学习路线图

很多初学者面对各种人工智能算法眼花缭乱，无所适从。

随着 Sklearn 模块库的完善，特别是神经网络算法模块的加入，目前学习人工智能、机器学习相对简单很多，初学者应直接从 Sklearn 开始学习，无须再借助其他模块库进行配套学习。

熟练掌握了 Sklearn 以后，就可以直接学习 TensorFlow 深度学习的知识内容。

以下是百度百科对于 TensorFlow 的介绍：

TensorFlow 是谷歌基于 DistBelief 进行研发的第二代人工智能学习系统，其命

名来源于本身的运行原理。

Tensor（张量）意味着 N 维数组，Flow（流）意味着基于数据流图的计算，TensorFlow 为张量从流图的一端流动到另一端的计算过程。TensorFlow 是将复杂的数据结构传输至人工智能神经网络进行分析和处理过程的系统。

TensorFlow 表达了高层次的机器学习计算，大幅简化了第一代系统，并且具备更好的灵活性和可延展性。TensorFlow 的一大亮点是支持异构设备分布式计算，能够在各个平台上自动运行模型，从手机、单个 CPU / GPU 到成百上千 GPU 卡组成的分布式系统。TensorFlow 支持 CNN、RNN 和 LSTM 算法，这都是目前在 Image、Speech 和 NLP 最流行的深度神经网络模型。

12.5.4 案例 12-5: MLP 神经网络算法

MLP（Multilayer Perceptron）神经网络算法，又称多层感知器神经网络算法，对应的函数名是 `MLPClassifier`。

多层感知器也叫人工神经网络（Artificial Neural Network，ANN），除了输入、输出层，中间可以有多个隐层，最简单的 MLP 只包含一个隐层，即三层的结构，如图 12-9 所示。

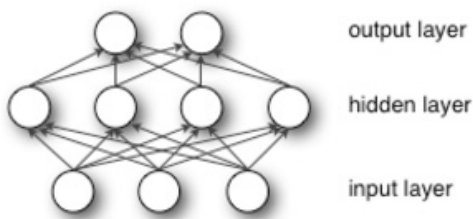


图 12-9 MLP 多层感知器神经网络算法

由图 12-9 可以看出，多层感知器层与各层之间是全连接的（全连接的意思就是：上一层的任何一个神经元与下一层的所有神经元都有连接）。多层感知器最底层是输入层，中间是隐藏层，最上面是输出层。

案例 12-5 的文件名是 `zai305_mx_MLP.py`，介绍 MLP 神经网络算法，函数位于 `neural_network` 神经网络模块，函数名是 `MLPClassifier`，函数接口是：

```
MLPClassifier(hidden_layer_sizes=(100, ), activation='relu',
solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant',
learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True,
random_state=None, tol=0.0001, verbose=False, warm_start=False,
momentum=0.9, nesterovs_momentum=True, early_stopping=False,
validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
```

案例 12-5 的核心是第 2 组的建模代码：

```
#2
print('\n2# 建模')
mx =zai.mx_MLP (x_train.values,y_train.values)
```

在本案例中，第 2 组代码使用 MLP 神经网络算法建立机器学习模型，但是并没有直接调用 Sklearn 模块库中的 MLPClassifier 函数，而是通过 ztop_ai 极宽智能模块库的函数接口间接进行调用，对应的函数代码是：

```
# MLP 神经网络算法
def mx_MLP(train_x, train_y):
    #mx = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_
layer_sizes=(5, 2), random_state=1)
    mx = MLPClassifier()
    mx.fit(train_x, train_y)
    return mx
```

MLP 神经网络算法位于 neural_network 神经网络模块，函数接口前面已经介绍过了，在此不再赘述。

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

有趣的是，MLP 神经网络算法每次运行的结果都有所不同，读者可以多运行几次试一下，对应的输出信息是：

```
4# df9
      x1  x2  x3  x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1         1         1         1
34  5.8  2.8  5.1  2.4         1         1         1
35  5.3  3.7  1.5  0.2         2         2         2
36  5.5  2.3  4.0  1.3         3         3         3
```



```
37 5.2 3.4 1.4 0.2          2          2          2
```

再一次运行的结果是：

```
4# df9
      x1  x2  x3  x4  y_predsr  y_test  y_pred
33 6.4  2.8  5.6  2.1          1          1          1
34 5.8  2.8  5.1  2.4          1          1          1
35 5.3  3.7  1.5  0.2          2          2          2
36 5.5  2.3  4.0  1.3          3          3          3
37 5.2  3.4  1.4  0.2          2          2          2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

因为 MLP 神经网络算法每次运行的结果不一定相同，所以准确度测试信息是：

```
5# mx:mx_sum,kok:89.47%
```

还有：

```
5# mx:mx_sum,kok:94.74%
```

这种动态的结果可能是 MLP 神经网络算法内部采用了不同的随机种子，以及计算流程的不同，从而造成最终的结果有误差。

另外，案例 12-5 的核心程序是第 2 组的建模代码：

```
#2
print('\n2# 建模')
mx =zai.mx_MLP (x_train.values,y_train.values)
```

可以发现，神经网络算法与其他的机器学习算法相比，无论是算法理论，还是具体实现都完全不同，但由于我们采用的都是 ztop_ai 极宽智能模块库的统一接口，使用完全一样，所以没有任何额外的学习成本。

12.5.5 案例 12-6: MLP_reg 神经网络回归算法

案例 12-6 的文件名是 zai306_mx_MLP_reg.py，介绍 MLP_reg 神经网络回归算法（又称多层感知器神经网络回归算法），函数位于 neural_network 神经网络模块，函数名是 MLPRegressor，函数接口是：

```
MLPRegressor(hidden_layer_sizes=(100, ), activation='relu',
```

```
solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant',
learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True,
random_state=None, tol=0.0001, verbose=False, warm_start=False,
momentum=0.9, nesterovs_momentum=True, early_stopping=False,
validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon= 1e-08)
```

案例 12-6 的核心是第 2 组的建模代码：

```
#2
print('\n2# 建模')
mx =zai.mx_MLP_reg(x_train.values,y_train.values)
```

在本案例中，第 2 组代码使用 MLP_reg 神经网络回归算法建立机器学习模型，但是并没有直接调用 Sklearn 模块库中的 MLPRegressor 函数，而是通过 ztop_ai 极宽智能模块库的函数接口间接进行调用，对应的函数代码是：

```
# MLP 神经网络回归算法
def mx_MLP_reg(train_x, train_y):
    #mx = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_
layer_sizes=(5, 2), random_state=1)
    mx = MLPRegressor()
    mx.fit(train_x, train_y)
    return mx
```

MLP_reg 神经网络回归算法位于 neural_network 神经网络模块，函数接口在前面已经介绍过了，在此不再赘述。

第 4 组代码，保存数据结果并显示相关信息：

```
#4
df9.to_csv('tmp/iris_9.csv',index=False)
print('\n4# df9')
print(df9.tail())
```

有趣的是，MLP_reg 神经网络回归算法每次运行的结果也有所不同，读者可以多运行几次试一下，对应的输出信息是：

```
4# df9
      x1  x2  x3  x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1  2.470013      1      2
34  5.8  2.8  5.1  2.4  2.393832      1      2
35  5.3  3.7  1.5  0.2  1.601259      2      2
36  5.5  2.3  4.0  1.3  1.969691      3      2
37  5.2  3.4  1.4  0.2  1.572605      2      2
```

再一次运行的结果是：

```
4# df9
      x1  x2  x3  x4  y_predsr  y_test  y_pred
33  6.4  2.8  5.6  2.1  2.307966      1      2
34  5.8  2.8  5.1  2.4  2.349912      1      2
35  5.3  3.7  1.5  0.2  1.898731      2      2
36  5.5  2.3  4.0  1.3  1.846610      3      2
37  5.2  3.4  1.4  0.2  1.809671      2      2
```

第 5 组代码，检验测试结果：

```
#5
dacc=zai.ai_acc_xed(df9,1,False)
print('\n5# mx:mx_sum,kok:{0:.2f}%'.format(dacc))
```

因为 MLP_reg 神经网络回归算法每次运行的结果不一定相同，所以准确度测试信息是：

```
5# mx:mx_sum,kok:34.21%
```

还有：

```
5# mx:mx_sum,kok:28.95%
```

这种动态的结果，也可能是神经网络回归算法内部采用了不同的随机种子，引发了计算方式的不同，从而造成最终的结果有误差。

28.95%~34.21%的准确度的确有点低，不过这正好是人工三选一的随机概率，这其实也是一个很有趣的课题。

虽然这个算法的准确度较低，但如果该准确度真的是神经网络算法造成的，那么在人性化模拟方面，也是非常成功的。

13

第 13 章

机器学习组合算法

前面几章，我们已经学习了多种常用的机器学习算法，本章将介绍机器学习的组合算法，就是集成多种不同的机器学习算法，从而提高预测结果的准确度。

13.1 CCPP数据集

在介绍机器学习组合算法之前，介绍一个新的经典数据集 CCPP 数据集。CCPP 是英文 Combined Cycle Power Plant 的缩写，是联合循环电厂的意思。

CCPP 数据集是源自企业的真实数据，是美国联合循环电厂 2006—2011 年超过 6 年的满负荷工作记录，共包含 9568 条数据，是以小时为单位的检测数据，每条数据均包括以下字段。

- AT: 温度。
- V: 压力。
- AP: 湿度
- RH: 压强。
- PE: 电力输出。

我们采用人工智能、机器算法的目的，就是根据前面四个字段即 AT、V、AP、RH 来建立机器学习模型，预测 PE 的数值。

CCPP 数据集也是美国加州大学欧文分校（University of California, Irvine，简称

UC Irvine 或 UCI) 机器学习公开课的课件数据。

CCPP 数据集原始文档是 Excel 格式, 我们已经改为 CSV 格式, 以便于 Pandas 处理, 文件名是 dat/ccpp.csv。

在 Sklearn 模块内部已经内置了大量的数据集, 包括前面介绍的 Iris 爱丽丝经典数据集。之所以使用外部数据集, 而不直接调用 Sklearn 模块的内部数据集, 是因为该模块问世较早, 当时还没有 Pandas 数据分析模块, 所以数据集使用的是 Numpy 格式, 使用起来很不方便。

此外, 使用 CCPP 数据集取代爱丽丝数据集还有以下因素。

- 爱丽丝数据集规模太小, 只有 150 条数据, 在性能测试方面无法体现各种机器学习算法的差异。
- CCPP 数据集有近万条数据, 规模适中, 既便于学习, 又可以反映出各种机器学习算法的性能差距, 如准确度、运行时间等。
- 爱丽丝数据集预测的结果是分类数据, CCPP 数据集的预测结果 PE 独立输出具体的数值, 这是机器学习预测结果的两种模式, 两个数据集刚好可以互补。

13.1.1 案例 13-1: CCPP 数据集

案例 13-1 的文件名是 zai401_ccpp01.py, 介绍 CCPP 数据集的基本情况。

案例 13-1 很简单, 主要代码如下:

```
fss='dat/ccpp.csv'
df=pd.read_csv(fss,index_col=False)
print('\n#1 df')
print(df.tail())
print(df.describe())
```

运行结果如下:

```
#1 df
      AT      V      AP      RH      PE
9563 15.12  48.92 1011.80  72.93  462.59
9564 33.41  77.95 1010.30  59.72  432.90
9565 15.99  43.34 1014.20  78.66  465.96
9566 17.65  59.87 1018.58  94.65  450.93
9567 23.68  51.30 1011.86  71.24  451.67
```

	AT	V	AP	RH	PE
count	9568.000000	9568.000000	9568.000000	9568.000000	9568.000000
mean	19.651231	54.305804	1013.259078	73.308978	454.365009
std	7.452473	12.707893	5.938784	14.600269	17.066995
min	1.810000	25.360000	992.890000	25.560000	420.260000
25%	13.510000	41.740000	1009.100000	63.327500	439.750000
50%	20.345000	52.080000	1012.940000	74.975000	451.550000
75%	25.720000	66.540000	1017.260000	84.830000	468.430000
max	37.110000	81.560000	1033.300000	100.160000	495.760000

由输出结果可以看出：

- CCpp 数据集共有 9568 条记录。
- AT 温度的平均数值是 19.65，最小值是 1.81，最大值是 37.11，中位数是 20.34，方差值是 7.45。
- V 压力的平均数值是 54.3，最小值是 25.36，最大值是 81.56，中位数是 52.08，方差值是 12.7。
- AP 湿度的平均数值是 1013.25，最小值是 992.89，最大值是 1033.3，中位数是 1012.94，方差值是 5.94。
- RH 压强的平均数值是 73.3，最小值是 25.56，最大值是 100.16，中位数是 74.98，方差值是 14.6。
- 最终结果数据，PE 电力输出的平均数值是 454.37，最小值是 420.25，最大值是 495.76，中位数是 451.55，方差值是 17.07。

13.1.2 案例 13-2: CCpp 数据切割

因为 Sklearn 的数据切割函数每次切割的结果都可能不同，所以为了保持学习体验一致，我们采用统一的训练数据集和测试数据集。

案例 13-2 的文件名是 zai402_ccpp02.py，介绍 CCpp 数据集的切割算法，以及切割后的各个数据文件的基本情况。

案例 13-2 的程序很简单，下面分组进行介绍。

第 1 组、第 2 组代码读取 CCpp 数据文件并保存到 df 变量，然后输出数据尾部信息：

```
#1
fss='dat/ccpp.csv'
df=pd.read_csv(fss,index_col=False)

#2
print('\n2# df')
print(df.tail())
```

对应的输出信息是：

```
2# df
      AT      V      AP      RH      PE
9563 15.12  48.92 1011.80  72.93  462.59
9564 33.41  77.95 1010.30  59.72  432.90
9565 15.99  43.34 1014.20  78.66  465.96
9566 17.65  59.87 1018.58  94.65  450.93
9567 23.68  51.30 1011.86  71.24  451.67
```

第 2 组代码是程序的核心所在，设置相关参数，并调用 `ai_data_cut` 函数切割数据：

```
#3
xlst,ysgn=['AT','V','AP','RH'],'PE'
ftg0='tmp/ccpp_'
ai_data_cut(df,xlst,ysgn,ftg0,True)
```

注意代码当中的 `xlst`、`ysgn` 是数据列的名称。

对应的输出结果如下：

```
tmp/ccpp_xtrain.csv
tmp/ccpp_xtest.csv
tmp/ccpp_ytrain.csv
tmp/ccpp_ytest.csv

x_train
      AT      V      AP      RH
2895 29.39  71.14 1010.97  53.88
7813 25.65  78.92 1010.83  86.56
905  10.22  39.64 1010.72  63.05
5192 20.32  44.60 1015.16  36.35
235  17.37  41.23  998.79  68.44
```

```
x_test
      AT      V      AP      RH
3854  26.34  69.45  1013.87  54.15
4785  25.06  64.63  1020.66  54.93
6406   8.34  40.96  1023.28  89.45
175   26.69  70.36  1006.82  68.29
7687  17.46  53.29  1018.07  91.01
```

```
y_train
2895    430.50
7813    434.78
905     477.22
5192    460.21
235     461.08
Name: PE, dtype: float64
```

```
y_test
3854    438.50
4785    446.57
6406    483.92
175     435.04
7687    458.06
Name: PE, dtype: float64
```

ok!

`ai_data_cut` 数据切割函数是位于 `ztop_ai` 极宽机器学习模块中的工具函数，为了便于学习，我们将该函数复制到主程序。

13.1.3 数据切割函数

`ai_data_cut` 数据切割函数代码如下：

```
def ai_data_cut(df,xlst,ysgn,ftg0,fgPr=False):
    x,y= df[xlst],df[ysgn]
    x_train, x_test, y_train, y_test = train_test_split(x, y,
random_state=1)
```



```

#
fss=ftg0+'xtrain.csv';x_train.to_csv(fss,index=False);print(fss)
fss=ftg0+'xtest.csv';x_test.to_csv(fss,index=False);print(fss)
fss=ftg0+'ytrain.csv';y_train.to_csv(fss,index=False,header=True);
print(fss)
fss=ftg0+'ytest.csv';y_test.to_csv(fss,index=False,
header=True);print(fss)
#
if fgPr:
    print('\nx_train');print(x_train.tail())
    print('\nx_test');print(x_test.tail())
    print('\ny_train');print(y_train.tail())
    print('\ny_test');print(y_test.tail())

```

函数代码很简单，就是调用 Sklearn 模块的 `train_test_split` 数据分割函数，把原始数据切割为以下 4 组。

- 训练数据集 `x_train`，文件名是 `ccpp_xtrain.csv`。
- 训练数据集对应的答案数据集 `y_train`，文件名是 `ccpp_ytrain.csv`。
- 测试数据集 `x_test`，文件名是 `ccpp_xtest.csv`。
- 测试数据集对应的答案数据集 `y_test`，文件名是 `ccpp_ytest.csv`。

以上数据文件在案例中输出到 `tmp` 目录，我们在 `dat` 目录中也复制了一份，以便于后面的案例学习。

13.1.4 案例 13-3：读取 CCPP 数据集

案例 13-3 的文件名是 `zai403_ccpp03.py`，介绍读取切割后的 CCPP 数据集，下面分组进行说明。

第 1 组代码，设置文件名前缀：

```

#1
fsr0='dat/ccpp_'
print('#2,',fsr0)

```

代码很简单，设置文件名前缀，调用 `ztop_ai` 极宽机器学习模块中的 `ai_dat_rd` 数据读取函数，读取所需的数据，并保存到对应的各个变量。

需要注意的是，机器学习所需的数据文件的文件名都是统一的前缀，后面的字

母分别代表各种数据，需要预先准备好。

- 训练数据集 `x_train`，文件名是 `xxx_xtrain.csv`。
- 训练数据集对应的答案数据集 `y_train`，文件名是 `xxx_ytrain.csv`。
- 测试数据集 `x_test`，文件名是 `xxx_xtest.csv`。
- 测试数据集对应的答案数据集 `y_test`，文件名是 `xxx_ytest.csv`。

如果调用 `ai_data_cut` 程序切割数据，会自动生成相应的数据文件。

第 1 组程序对应的文件名在第 2 组输出信息的开头部分，内容如下：

```
dat/ccpp_xtrain.csv
dat/ccpp_xtest.csv
dat/ccpp_ytrain.csv
dat/ccpp_ytest.csv
```

第 2 组、第 3 组代码分别设置 `k0=1`、`k0=10`，调用 `ztop_ai` 极宽机器学习模块中的 `ai_dat_rd` 数据读取函数来读取所需的数据，并保存到对应的各个变量，代码如下：

```
#2
k0=1
print('\n#2,k0',k0)
x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0,k0,True)

#3
k0=10
print('\n#3,k0',k0)
x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0,k0,True)
```

两组代码的输出数据大部分相同，唯一有差别的是两组 `y` 字头的答案数据。

- 训练数据集对应的答案数据集为 `y_train`。
- 测试数据集对应的答案数据集为 `y_test`。

对应的输出信息如下：

<code>y_train</code>	<code>y_train</code>
PE	PE
7171 430	7171 4305
7172 435	7172 4348
7173 477	7173 4772
7174 460	7174 4602
7175 461	7175 4611

y_test

PE

2387 438

2388 447

2389 484

2390 435

2391 458

第 2 组输出信息, k0=1

y_test

PE

2387 4385

2388 4466

2389 4839

2390 4350

2391 4581

第 3 组输出信息, k0=10

由输出信息可以看出, 第 3 组的输出数据大约是第 2 组的 10 倍, 具体原因可以参看 ai_dat_rd 数据读取函数的源码。

13.1.5 数据读取函数

案例 13-3 中调用了 zai.ai_dat_rd 数据读取函数, 代码如下:

```
def ai_dat_rd(fsr0,k0=1,fgPr=False):
    #1
    fss=fsr0+'xtrain.csv';x_train=pd.read_csv(fss,index_col=
False);print(fss)
    fss=fsr0+'xtest.csv';x_test=pd.read_csv(fss,index_col=
False);print(fss)
    fss=fsr0+'ytrain.csv';y_train=pd.read_csv(fss,index_col=
False);print(fss)
    fss=fsr0+'ytest.csv';y_test=pd.read_csv(fss,index_col=
False);print(fss)
    #2
    ysgn=y_train.columns[0];#print('y',ysgn)
    y_train[ysgn]=round(y_train[ysgn]*k0).astype(int)
    y_test[ysgn]=round(y_test[ysgn]*k0).astype(int)
    #3
    if fgPr:
        print('\nx_train');print(x_train.tail())
        print('\nx_test');print(x_test.tail())
        print('\ny_train');print(y_train.tail())
        print('\ny_test');print(y_test.tail())
    #4
```

```
return x_train, x_test, y_train, y_test
```

函数很简单，下面分组进行说明。

- 第 1 组代码：读取各个数据文件，并保存到相应的变量。
- 第 2 组代码：需要注意，把 y 字头的答案数据改为整数类型，个别原始数据值较小，需要乘以 k0 进行数值放大，这是因为部分机器学习算法的答案数据只支持整数格式，很多初学者都在此处遇到瓶颈。
- 第 3 组代码：根据 fgPr 布尔变量输出相关的变量数据信息。
- 第 4 组代码：需要注意，函数的返回值是多组，注意各个变量的名称和次序，一般是直接复制到调用主程序。

13.2 机器学习统一接口函数

在前面的案例中，我们介绍的各种机器学习算法都位于 ztop_ai 极宽机器学习模块中，而且函数名称都是 mx_xxx 格式，调用的数据接口也是一致的。

这样的二次封装，虽然大大简化了机器学习算法的调用过程，降低了学习难度，但作为机器学习的组合调用还不够，需要再追加一个标准的机器学习调用接口函数。

13.2.1 案例 13-4：机器学习统一接口

案例 13-4 的文件名是 zai404_mx_01.py，介绍使用标准的接口，统一调用机器学习算法函数建模、分析及得出预测结果，下面分组逐一进行介绍。

第 1 组代码，设置有关参数，并读取已经分割好的训练数据、测试数据和对应的答案数据，其中 xlst、ysgn 是数据源的数据字段名称，相关代码如下：

```
#1
fsr0='dat/ccpp_'
print('#1',fsr0)
x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0)
```

对应的输出信息是：

```
#1 dat/ccpp_
dat/ccpp_xtrain.csv
dat/ccpp_xtest.csv
```

```
dat/ccpp_ytrain.csv
dat/ccpp_ytest.csv
```

输出信息包括文件名前缀和各组数据的文件名。

第 2 组代码, 设置机器学习函数名称, 并调用统一的接口函数 `mx_fun010`, 相关代码如下:

```
#2
print('\n#2,mx_line')
#mx_fun=zai.mx_line
funSgn='line'
tim0=arrow.now()
dacc,df9=mx_fun010(funSgn,x_train, x_test, y_train, y_test, 5,False)
tn=zt.timNSec('',tim0,True)
```

第 2 组代码设置的机器学习函数是线性回归函数, 并计算运行时间, 其中函数名称是通过 `funSgn` 字符串代码传递的。

对应的结果数据如下:

```
#2,mx_line
@mx:mx_sum,kok:99.96%
0.01 s, 09:11:38 ,t0, 09:11:38
```

第 3 组代码与第 2 组代码类似, 但采用的是逻辑回归算法, 此外, 调用时最后一个输入参数 `fgDebug=True`, 采用调试模式可以输出更多信息, 相关代码如下:

```
#3
print('\n#3,mx_log')
funSgn='log'
tim0=arrow.now()
dacc,df9=mx_fun010(funSgn,x_train, x_test, y_train, y_test,
5,False,True)
tn=zt.timNSec('',tim0,True)
```

对应的输出信息是:

```
#3,mx_log
ai_acc_xed
      AT  V   AP  RH  y_test  y_pred  ysub
ysub2      y_test_div      ysubk
0   17.80   43.72  1008.71  78.50   459 466 -7
7        459.0          1.525054
1   29.60   71.14  1011.46  52.69   431 436 -5
```

```

5      431.0      1.160093
2    11.06    36.71    1021.67 80.44    474 474 0
0      474.0      0.000000
3    30.06    67.25    1017.63 53.59    435 440 -5
5      435.0      1.149425
4    19.88    47.03    1012.27 91.99    456 466 -10
10     456.0      2.192982

```

```

test,2392,npred,2392,dsum,2384
acc-kok: 99.67%, MAE:5.04, MSE:42.68, RMSE:6.53
@fun name: mx_log
@mx:mx_sum,kok:99.67%
0.99 s, 09:11:39 ,t0, 09:11:38

```

在调试模式下，会输出较多信息。

在最后的結果中：

- 逻辑回归算法的准确度是 99.67%，耗时 1 秒，但因为是调试模式，输出信息过多，所以无法反映真实的运行时间；
- 线性回归算法的准确度是 99.96%，耗时 0.01 秒。

13.2.2 统一接口函数

在案例中使用的统一的接口函数 `zai.mx_fun010` 原本位于 `ztop_ai` 极宽机器学习模块中，为了读者学习方便，我们复制了一份到主程序。

下面对 `zai.mx_fun010` 函数分组进行讲解。

`zai.mx_fun010` 函数的接口定义如下：

```

def mx_fun010(funSgn,x_train, x_test, y_train, y_test,yk0=5,
fgInt=False,fgDebug=False):

```

其中，各函数定义如下。

- `funSgn`: 字符串格式，机器学习函数名称代码，实际调用的函数是 `ztop_ai` 极宽机器学习模块中二次封装的函数，函数名一般是 `mx_xxx`。
- `x_train,x_test,y_train,y_test`: 学习和测试数据集，使用 Pandas 的 `DataFrame` 格式。
- `yk0=5`: 结果数据误差 k 值，默认是 5，表示 5%；整数模式设置为 1。
- `fgInt=False`: 整数结果模式，默认是 `False`。

- fgDebug=False: 调试模式, 默认是 False。

第 1 组代码, 设置结果变量 df9 和机器学习函数 mx_fun, 并且调用该函数进行机器学习, 构建算法模型, 并保存到变量模型。

```
#1
df9=x_test.copy()
mx_fun=mxfunSgn[funSgn]
mx =mx_fun(x_train.values,y_train.values)
```

第 2 组代码, 调用 mx 模型变量内置的 predict 预测函数, 生成预测结果并保存到变量 y_pred, df9 的相关参数如下:

```
#2
y_pred = mx.predict(x_test.values)
df9['y_test'],df9['y_pred']=y_test,y_pred
```

第 3 组代码, 默认是非整数模式, fgInt 结果数据为整数模式才执行, 把结果变量 df9 中的数据格式转换为整数格式。

Iris 爱丽丝数据集和一些分类算法, 其结果都需要采用整数模式, CCPP 数据集的案例输入时需要对结果数据进行整数化处理, 但结果无须采用整数格式。

```
#3
if fgInt:
    df9['y_preds']=df9['y_pred']
    df9['y_pred']=round(df9['y_preds']).astype(int)
```

第 4 组代码, 调用自定义的 ai_acc_xed 效果评估函数, 生成结果数据, 注意 fgDebug 调试模式参数也会传递至 ai_acc_xed 效果评估函数。

```
#4
dacc=ai_acc_xed(df9,yk0,fgDebug)
```

第 5 组代码, 如果调试模式变量 fgDebug=True, 则输出结果数据变量 df9 的尾部数据, 并把 df9 保存到文件 tmp/df9_pred.csv。

```
#5
if fgDebug:
    #print(df9.head())
    print('@fun name:',mx_fun.__name__)
    df9.to_csv('tmp/df9_pred.csv');
```

第 6 组代码, 输出评估结果, 返回 dacc 评估数值和结果变量 df9, 其中 dacc 评估数值采用百分比数据。

```
#6
```

```
print('@mx:mx_sum,kok:{0:.2f}%'.format(dacc))
return dacc,df9
```

13.2.3 机器学习算法代码

在 `zai.mx_fun010` 函数中使用的 `funSgn` 字符串变量是机器学习函数的代码名称，定义在 `ztop_ai` 极宽机器学习模块中，位置在 `mx_xxx` 函数定义之后，`mx_fun010` 函数定义前的相关代码如下：

```
#-----mx.fun.sgn

mxfunLst=['line','log','bayes','knn','forest','dtree','gbdt','svm','s
vmcr','mlp','mlpreg']
mxfunSgn={'line':mx_line,
          'log':mx_log,
          'bayes':mx_bayes,
          'knn':mx_knn,
          'forest':mx_forest,
          'dtree':mx_dtrees,
          'gbdt':mx_GBDT,
          'svm':mx_svm,
          'svmcr':mx_svm_cross,
          'mlp':mx_MLP,
          'mlpreg':mx_MLP_reg
        }
```

其中：

- `mxfunLst` 是列表格式，是各个机器学习算法函数的缩写代码；
- `mxfunSgn` 是字典格式，是各个机器学习算法函数的缩写代码和对应的函数名称。

`mxfunLst` 和 `mxfunSgn` 没有按一般的参数定义放在模块前部，而是放在模块中部这个位置，是因为字典中的函数名称必须先对应才能设置，不然会出现错误。

`mxfunLst` 函数列表和 `mxfunSgn` 字典变量，随着 `ztop_ai` 极宽机器学习模块的升级，会不断进行优化调整。

13.2.4 效果评估函数

`ai_acc_xed` 效果评估函数位于 `ztop_ai` 极宽机器学习模块中，主要用于评估机器学习算法函数的效果，下面分组逐一进行讲解。

`ai_acc_xed` 效果评估函数的接口定义如下：

```
def ai_acc_xed(df9,ky0=5,fgDebug=True):
```

其中：

- `df9`，Pandas 的 `DataFrame` 格式，结果数据保存变量；
- `ky0`，结果数据误差 `k` 值，默认是 5，表示 5%；整数模式设置为 1；
- `fgDebug`，调试模式变量，默认为 `False`。

第 1 组代码，设置预测数据与实际结果数据的绝对误差值，并保存到新增数据列 `ysub2`：

```
#1
ny_test,ny_pred=len(df9['y_test']),len(df9['y_pred'])
df9['ysub']=df9['y_test']-df9['y_pred']
df9['ysub2']=np.abs(df9['ysub'])
```

第 2 组代码如下：

```
#2
df9['y_test_div']=df9['y_test']
df9.loc[df9['y_test'] == 0, 'y_test_div'] =0.00001
df9['ysubk']=(df9['ysub2']/df9['y_test_div'])*100
dfk=df9[df9['ysubk']<ky0]
dsum=len(dfk['y_pred'])
dacc=dsum/ny_test*100
```

运行流程如下。

- 计算结果数据绝对差值 `ysub2` 与实际结果数据 `y_test` 之间的差距，结果为百分比，并保存在数据列 `ysubk`。
- 提取 `ysubk` 小于设定的误差值 `ky0` 的数据到变量 `dfk`。
- 计算变量 `dfk` 的长度，结果就是正确的预测结果数目，并保存在变量 `dsum`。
- 计算 `dsum` 与测试数据总数目 `ny_test` 的百分比，结果就是预测结果的准确度，并保存在变量 `dacc`。

第 3 组代码，如果调试模式变量 `fgDebug=True`，则输出相关数据，并调用 `Sklearn`

的测试模块 `Metrics` 进行进一步的测试：

```
#
#3
if fgDebug:
    print('\nai_acc_xed')
    print(df9.head())
    y_test,y_pred=df9['y_test'],df9['y_pred']
    print('\ntest,{0},npred,{1},dsum,{2}'.format(ny_test,
ny_pred,dsum))

    dmae=metrics.mean_absolute_error(y_test, y_pred)
    dmse=metrics.mean_squared_error(y_test, y_pred)
    drmse=np.sqrt(metrics.mean_squared_error(y_test, y_pred))
    print('acc-kok: {0:.2f}%, MAE:{1:.2f}, MSE:{2:.2f},
RMSE:{3:.2f}'.format(dacc,dmae,dmse,drmse))
```

其中对应的输出结果如下：

```
ai_acc_xed
   AT   V   AP   RH  y_test  y_pred  ysub  ysub2  y_test_div  ysubk
0 17.80 43.72 1008.71 78.50   459   466   -7    7    459.0  1.525054
1 29.60 71.14 1011.46 52.69   431  436   -5    5    431.0  1.160093
2 11.06 36.71 1021.67 80.44   474  474    0    0    474.0  0.000000
3 30.06 67.25 1017.63 53.59   435  440   -5    5    435.0  1.149425
4 19.88 47.03 1012.27 91.99   456  466  -10   10    456.0  2.192982

test,2392,npred,2392,dsum,1283
acc-kok: 53.64%, MAE:5.04, MSE:42.68, RMSE:6.53
```

13.2.5 常用评测指标

`ai_acc_xed` 效果评估函数中的 MAE、MSE、RMSE 都是统计术语，用于结果评测，定义分别如下。

- MAE: Mean Absolute Error，平均绝对误差。
- MSE: Mean Squared Error，均方差、方差，结果数据越接近于零，说明模型选择和拟合更好，数据预测也越成功。
- RMSE: Root Mean Squared Error，均方根、标准差，结果数据越接近于零，说

明模型选择和拟合更好，数据预测也越成功。

还有以下类似术语。

- **SSE (Sum of Squares due to Error)**: 和方差、误差平方和。SSE 越接近于零，说明模型选择和拟合更好，数据预测也越成功。MSE 与 RMSE 因为和 SSE 同出一宗，所以效果一样。
- **R-square**: 确定系数，通过数据的变化来表征一个拟合算法的好坏，正常取值范围为[0 1]，越接近 1，表明方程的变量对 y 的解释能力越强，算法模型对数据拟合也较好。
- **Adjusted R-square (Degree-of-freedom Adjusted Coefficient of Determination)**: 修正确定系数。
- **Precision**: 又称 P 指标，表示准确率，即检索出来的条目（比如文档、网页等）有多少是准确的。
- **Recal**: 又称 R 指标，表示召回率，即所有准确的条目有多少被检索出来了。
- **F 值 (F-Measure)**: 又称为 F-Score，F 值是正确率和召回率的调和平均值， $F \text{ 值} = \text{正确率} \times \text{召回率} \times 2 / (\text{正确率} + \text{召回率})$ 。
- **E 值**: 表示查准率 P 和查全率 R 的加权平均值，当其中一个为零时，E 值为 1。
- **AP (Average Precision)**: 平均正确率，表示不同查全率的点上的正确率的平均值。

13.3 批量调用机器学习算法

前面的有关案例介绍了各种机器学习算法和统一的接口，本节将通过机器学习算法的统一接口，对各种机器学习算法进行批量调用，同时测试各种算法、模型的准确度和速度。

13.3.1 案例 13-5: 批量调用

案例 13-5 的文件名是 zai405_mx_02.py，介绍如何批量调用各种机器学习算法。第 1 组代码，设置各种参数，读取训练和测试数据集。

```
#1
mxlst1=['line', 'knn', 'bayes']
```

```
mxlst2=['line','log','bayes','knn','forest','dtree','svm','mlp','mlpreg']
# 'gbd', 'svmcr'

fsr0='dat/ccpp_'
print('#1',fsr0)
x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0)
```

其中, mxlst1、mxlst2 是批量调用的机器学习字符串代码列表, 都是小写格式。这些标识符源自 ztop_ai 极宽机器学习模块中的 mxfunLst:

```
mxfunLst=['line','log','bayes','knn','forest','dtree','gbd', 'svm', 'svmcr', 'mlp', 'mlpreg']
```

随着 ztop_ai 版本的升级, 具体代码和标识符可能会有所变动, 以最新发布的软件代码为准。

注意, mxlst1、mxlst2 列表中各种标识符的位置、次序只是影响对应机器学习函数的调用次序, 对其他结果没有影响。

在 mxlst2 列表中没有 gbd、svmcr 两种算法的标识符, 是因为测试时这两种算法出现错误, 可能是 CCPP 数据不适合这两种算法。这也说明, 各种算法模型都有一定的局限性。

第 2 组代码, 使用 mxlst1 调用 zai.mx_funlst 批量调用函数:

```
#2
print('\n#2,mxlst1')
zai.mx_funlst(mxlst1,x_train, x_test, y_train, y_test)
```

对应的输出信息是:

```
#2,mxlst1

line
@mx:mx_sum,kok:99.96%
0.01 s, 09:18:05 ,t0, 09:18:05

knn
@mx:mx_sum,kok:99.50%
0.04 s, 09:18:05 ,t0, 09:18:05

bayes
@mx:mx_sum,kok:99.83%
```

```
0.02 s, 09:18:05 ,t0, 09:18:05
```

mxlst1 只有 Line（线性回归）、KNN（K 近邻）、Bayes（贝叶斯）3 种算法，函数运行速度都很快，在 0.01~0.04 秒，准确度也差不多，都在 99%左右。

第 3 组代码，使用 mxlst2 调用 zai.mx_funlst 批量调用函数：

```
#3
print('\n#3,mxlst2')
zai.mx_funlst(mxlst2,x_train, x_test, y_train, y_test,False)
```

对应的输出信息是：

```
#3,mxlst2

line
@mx:mx_sum,kok:99.96%
0.01 s, 09:18:05 ,t0, 09:18:05

log
@mx:mx_sum,kok:99.67%
0.96 s, 09:18:06 ,t0, 09:18:05

bayes
@mx:mx_sum,kok:99.83%
0.02 s, 09:18:06 ,t0, 09:18:06

knn
@mx:mx_sum,kok:99.50%
0.04 s, 09:18:06 ,t0, 09:18:06

forest
@mx:mx_sum,kok:99.67%
0.27 s, 09:18:06 ,t0, 09:18:06

dtree
@mx:mx_sum,kok:99.75%
0.07 s, 09:18:06 ,t0, 09:18:06

svm
@mx:mx_sum,kok:96.40%
```

```
29.65 s, 09:18:36 ,t0, 09:18:06
```

```
mlp
@mx:mx_sum,kok:99.75%
1.42 s, 09:18:37 ,t0, 09:18:36
```

```
mlpreg
y = column_or_1d(y, warn=True)
@mx:mx_sum,kok:99.96%
1.22 s, 09:18:38 ,t0, 09:18:37
```

在输出信息中，可以看到 SVM 向量机算法所需时间最久，将近 30 秒。MLP、MLPreg 神经网络算法和 Log 回归算法的时间也较长，在 1~1.4 秒，其他的机器学习算法一般在 1 秒之内。

在准确度方面：

- 除了 SVM 向量机算法准确度是 96.4%，其他算法准确度的都在 99%左右；
- SVM 向量机算法准确度较低，但并不代表 SVM 向量机算法效果最差，只是说明该算法不太适合 CCPP 数据集；
- 由于不少机器学习算法内部都采用了随机种子，所以每次运行的结果可能会有略微差异，但整体数值都差不多。

13.3.2 批量调用算法模型

在案例中，批量调用机器学习算法使用的是 `zai.mx_funlst` 批量调用函数，相关代码如下：

```
def mx_funlst(funlst,x_train, x_test, y_train, y_test,yk0=5,
fgInt=False):
    for funsgn in funlst:
        print('\n',funsgn)
        tim0=arrow.now()
        mx_fun010(funsgn,x_train, x_test, y_train, y_test,yk0, fgInt)
        zt.timNSec('',tim0,True)
```

由以上代码可以看出，有了统一调用接口函数 `mx_fun010` 后，批量调用机器学习算法其实非常简单，一个 for 循环即可完成。

13.4 一体化调用

下面介绍一个一体化的机器学习调用案例，从原始数据到直接分割数据，然后进行学习训练和结果预测，更加贴近实盘的调用模式。

13.4.1 案例 13-6：一体化调用

案例 13-6 的文件名是 `zai406_mx_03.py`，演示一体化调用机器学习算法，程序很简单，核心代码如下：

```
#1
xlst,ysgn=['AT','V','AP','RH'],'PE'
df=pd.read_csv('dat/ccpp.csv',index_col=False)

#2
print('\n#2,mx_line')
funsgn='line'
tim0=arrow.now()
zai.mx_fun_call(df,xlst,ysgn,funsgn)
tn=zt.timNSec('',tim0,True)
```

从以上代码可以看出，一体化调用代码更加简洁，因为省略了各个切割好的训练数据，以及读取数据的参数设置和加载。

核心代码只有一条，调用 `mx_fun_call` 函数：

```
zai.mx_fun_call(df,xlst,ysgn,funsgn)
```

可以看出，一体化调用的函数定义接口也简化了很多。

程序运行结果如下：

```
#2,mx_line

n_tran:, 7176 ,ntst, 2392 ,dimension:, 4 ,kbin, False

y_pred,预测
@mx:mx_sum,kok:99.96%
0.02 s, 10:05:23 ,t0, 10:05:23
```

13.4.2 一体化调用函数

一体化调用函数 `mx_fun_call` 位于 `ztop_ai` 极宽机器学习模块中，下面我们分组进行讲解。

函数定义与接口代码如下：

```
def mx_fun_call(df,xlst,ysgn,funSgn,yksiz=1,yk0=5,fgInt=False,
fgDebug=False):
```

其中，各字母含义如下。

- **df**: 数据源，可以是原始数据，也可以是归一化处理的衍生数据，不过必须全部是数字。
- **xlst**: 参数数据集字段名称，源自 **df** 的字段名。
- **ysgn**: 对应的答案字段名称。
- **funSgn**: 机器学习算法字符串代码。
- **yksiz**: 结果数据缩放比例，默认是 1。
- **ky0**: 结果数据误差 **k** 值，默认是 5，表示 5%；整数模式设置为 1。
- **fgInt**: 整数结果模式变量，默认为 `False`。
- **fgDebug**: 调试模式变量，默认为 `False`。

第 1 组代码，设置 **ysgn** 答案数据列的数据格式为整数，如果 **ysgn** 源数据过小，需要设置合适的 **yksiz** 参数进行调整：

```
#1
df[ysgn]=df[ysgn].astype(float)
df[ysgn]=round(df[ysgn]*yksiz).astype(int)
```

第 2 组代码，设置主数据集 **x** 和对应的答案数据集 **y**：

```
#2
x,y= df[xlst],df[ysgn]
```

第 3 组代码，分割数据为 **x_train**、**x_test**、**y_train** 和 **y_test**：

```
#3
x_train, x_test, y_train, y_test = train_test_split(x, y, random_
state=1)
num_train, num_feat = x_train.shape
num_test, num_feat = x_test.shape
print('\nn_train:',num_train,' ,ntst,' , num_test, ' ,dimension:',
',num_feat,',kbin,',fgInt)
```


第 4 组代码, 根据 `funSgn` 机器学习算法标识符, 设置对应的函数, 并调用建模, 结果保存在变量 `mx` 中:

```
#4
print('\ny_pred, 预测')
df9=x_test.copy()
mx_fun=mxfunSgn[funSgn]
mx =mx_fun(x_train.values,y_train.values)
```

第 5 组代码, 调用模型变量 `mx` 的内置 `predict` 预测函数, 生成预测数据 `y_pred`, 并保存到结果变量 `df9`:

```
#5
y_pred = mx.predict(x_test.values)
df9['y_test'],df9['y_pred']=y_test,y_pred
```

第 6 组代码, 如果整数答案模式 `fgInt=True`, 则把结果变量 `df9` 中的 `y_pred` 数据列改为整数格式:

```
#6
if fgInt:
    df9['y_preds_r']=df9['y_pred']
    df9['y_pred']=round(df9['y_preds_r']).astype(int)
```

第 7 组代码, 调用 `ai_acc_xed` 效果评测函数, 计算预测结果的准确度:

```
#7
dacc=ai_acc_xed(df9,yk0,fgDebug)
```

第 8 组代码, 如果调试模式 `fgDebug=True`, 则输出结果变量 `df9` 的部分数据, 并保存到文件 `tmp/df9_pred.csv`:

```
#8
if fgDebug:
    #print(df9.head())
    print('@fun name:',mx_fun.__name__)
    df.to_csv('tmp/df_sr.csv');
    df9.to_csv('tmp/df9_pred.csv');
```

第 9 组代码, 输出准确度数据, 返回结果数据 `dacc` 和 `df9`:

```
#9
print('@mx:mx_sum,kok:{0:.2f}%'.format(dacc))
return dacc,df9
```

13.5 模型预制与保存

在机器学习算法批量调用案例中，SVM 向量机算法所需时间最久，将近 30 秒。MLP 神经网络算法、MLPreg 神经网络回归算法和 Log 回归算法的时间大约 1~1.4 秒，相对于 SVM 向量机算法的 30 秒，算是比较快的了。

CCPP 数据集只有 1 万条数据，在实盘中，单只股票每天的 ticks 数据就超过 1 万条，在足彩数据中，gid 基本比赛数据有 7 万条，总的赔率数据有近 230 万条。

在实盘中，往往耗时最多的是机器学习构建算法模型，而真正预测的数据却不多，也无须很长的计算时间。

建模使用的是历史数据，经常是数万条记录，甚至百万条、千万条数据，某些大数据项目更是以亿单位为起点。

实盘预测往往只是采用实时数据，一般都是当天的最新数据，配合一些其他数据，比如均线数据，所以数据量并不大。

以往，我们的案例都是把建模和预测放在一起，如果能够分开，实盘分析无须建模，直接预测，则速度至少可以快 10 倍以上。

Python 语言的 Pickle 模块支持二进制的数据和程序一体化储存，是非常理想的机器学习模型储存工具。

在 Sklearn 模块中，内置了高效的模型持久化模块 Joblib，将模型保存至硬盘，并且从硬盘加载，效率更高。

使用时，代码也非常简单：

```
from sklearn.externals import joblib
#假设mx是一种机器学习的模型
joblib.dump(mx, 'mx.pkl')
mx = joblib.load('mx.pkl')
```

Joblib 使用的文件格式是 PKL，本质上也是 Pickle 模块的一种文件保存格式，而且 Pickle 也可以直接读取 PKL 文件。

13.5.1 案例 13-7：储存算法模型

案例 13-7 的文件名是 zai407_mx_04.py，介绍使用 Sklearn 模块中的 Joblib 函数储存并读取机器学习算法模型，下面分组逐一进行介绍。

第 1 组代码，设置参数，读取训练和测试数据集，以及 mx 模型文件名。

```
#1
fsr0='dat/ccpp_'
print('#1',fsr0)
xlst,ysgn=['AT','V','AP','RH'],'PE'
x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0)
funSgn,ftg='svm','tmp/ccpp_svm.pkl'
```

第 2 组代码，通过 funSgn 调用 zai.ai_f_mxWr 函数，对输入数据运行机器学习算法，并保存到文件：

```
#2
print('\n#2,mx_svm.wr')
tim0=arrow.now()
zai.ai_f_mxWr(ftg,funSgn,x_train, y_train)
tn=zt.timNSec('',tim0,True)
```

对应的输出信息是：

```
#2,mx_svm.wr
27.88 s, 11:09:44 ,t0, 11:09:16
```

在案例中使用的是 SVM 向量机模型，该模型耗时较长，以前的案例调用大约是 30 秒，此处训练时间大约是 27.88 秒。

第 3 组代码，读取 mx 模型数据：

```
#3
tim0=arrow.now()
print('\n#3,mx_svm.rd')
mx = joblib.load(ftg)
tn=zt.timNSec('',tim0,True)
```

对应的输出信息是：

```
#3,mx_svm.rd
0.01 s, 11:09:44 ,t0, 11:09:44
```

读取数据很快，才 0.01 秒。

第 4 组代码，使用读取的 mx 模型数据进行预测分析：

```
#4
print('\n#4,mx_svm')
tim0=arrow.now()
zai.mx_fun8mx(mx,x_test,y_test)
```

```
tn=zt.timNSec('',tim0,True)
```

对应的输出信息是：

```
#4,mx_svm
@mx:mx_sum,kok:96.40%
1.83 s, 11:09:46 ,t0, 11:09:44
```

准确度是 96.4%，运行时间是 1.83 秒，比以前的 30 秒快了近 20 倍。

13.5.2 模型保存函数

模型保存函数 `ai_f_mxWr` 位于 `ztop_ai` 极宽机器学习模块中，代码如下：

```
def ai_f_mxWr(ftg,funSgn,x_train, y_train):
    #1
    mx_fun=mxfunSgn[funSgn]
    mx =mx_fun(x_train.values,y_train.values)
    #2
    joblib.dump(mx,ftg)
```

以上代码非常简单，读者自己解读即可。

13.5.3 模型预测函数

模型预测函数 `mx_fun8mx` 通过预先保存或者生成的算法模型分析输入数据，函数位于 `ztop_ai` 极宽机器学习模块中，代码如下：

```
def mx_fun8mx(mx,x_test,y_test,yk0=5,fgInt=False,fgDebug=False):
    #1
    df9=x_test.copy()
    #mx=....
    #2
    y_pred = mx.predict(x_test.values)
    df9['y_test'],df9['y_pred']=y_test,y_pred
    #3
    if fgInt:
        df9['y_predsr']=df9['y_pred']
        df9['y_pred']=round(df9['y_predsr']).astype(int)
```

```

#4
dacc=ai_acc_xed(df9,yk0,fgDebug)
#5
if fgDebug:
    #print(df9.head())
    print('@fun name:',mx_fun.__name__)
    df9.to_csv('tmp/df9_pred.csv');
#
#6
print('@mx:mx_sum,kok:{0:.2f}%'.format(dacc))
return dacc,df9

```

以上函数代码看起来较长，但其实不过是 `mx_fun010` 函数后面的一部分，读者参考函数 `mx_fun010` 的介绍即可理解以上代码。

事实上，模型保存函数 `ai_f_mxWr` 和模型预测函数 `mx_fun8mx`，都是从 `mx_fun010` 函数中分割出来的，有兴趣的读者可以对比分析一下。

13.5.4 案例 13-8：批量储存算法模型

需要说明的是，`Sklearn` 模块保存的机器学习算法模型，不同的算法模型保存的模型不同，即使是同一个算法，不同的训练数据，保存的模型也不同。

例如 `CCPP` 数据集，如果每次调用的数据分割函数，动态生成训练数据和测试数据，那么每次的训练模型都要采用不同的文件名进行保存，否则会出现混淆。

为了提高效率，我们需要批量储存算法模型，案例 13-8 就是介绍批量储存不同的算法模型的。

案例 13-8 的文件名是 `zai408_mx_05.py`，程序很简单，核心代码如下：

```

#1
fsr0='dat/ccpp_'
print('#1',fsr0)
mxlst=['line','log','bayes','knn']
x_train, x_test, y_train, y_test=zai.ai_dat_rd(fsr0)
ftg0='tmp/ccpp_'

```

```
#2
print('\n#2,mx_svm.wr')
zai.ai_f_mxWrlst(ftg0,mxlst,x_train, y_train)
```

以上程序除了最后一行代码，其他的都已经介绍过，最后一行代码如下：

```
zai.ai_f_mxWrlst(ftg0,mxlst,x_train, y_train)
```

这行代码是本案例的核心，调用 `ztop_ai` 极宽机器学习模块中的 `ai_f_mxWrlst` 函数，批量储存 CCPP 数据集的相关模型代码，案例中使用了 Line（线性回归）、Log（逻辑回归）、Bayes（贝叶斯）和 KNN（K 近邻）4 种不同的机器学习算法。

对应的输出信息如下：

```
#2,mx_svm.wr

tmp/ccpp_line.pkl
0.0 s, 11:36:22 ,t0, 11:36:22

tmp/ccpp_log.pkl
0.97 s, 11:36:23 ,t0, 11:36:22

tmp/ccpp_bayes.pkl
0.01 s, 11:36:23 ,t0, 11:36:23

tmp/ccpp_knn.pkl
0.01s, 11:36:23 ,t0, 11:36:23
```

几种算法的运行时间都差不多，除了 Log 逻辑回归算法用了 1 秒，其他基本上都是零点零几秒，除了以上输出信息，程序还在 `tmp` 目录下生成了多个对应的 PKL 算法模型文件，如图 13-1 所示。



名称	大小	修改日期	类型
ccpp_bayes.pkl	7 KB	2017/2/19 11:36	PKL 文件
ccpp_knn.pkl	559 KB	2017/2/19 11:36	PKL 文件
ccpp_line.pkl	1 KB	2017/2/19 11:36	PKL 文件
ccpp_log.pkl	5 KB	2017/2/19 11:36	PKL 文件
ccpp_svm.pkl	8,519 KB	2017/2/19 11:09	PKL 文件

图 13-1 机器学习算法模型文件

由图 13-1 可以看出，各种算法模型的文件大小差别很大，SVM 向量机的模型文件有 8MB 多，而 Line 线性回归算法的模型文件只有 1KB 大小。

需要注意的是，不同规模的训练数据其模型文件的大小也是不同的，后面的磁盘案例会进一步介绍。

13.5.5 批量模型储存函数

ai_f_mxWrlst 批量储存函数位于 ztop_ai 极宽机器学习模块中，函数代码如下：

```
def ai_f_mxWrlst(ftg0,funlst,x_train, y_train):
    for funSgn in funlst:
        ftg=ftg0+funSgn+'.pkl'
        print('\n',ftg)
        tim0=arrow.now()
        ai_f_mxWr(ftg,funSgn,x_train, y_train)
        zt.timNSec('',tim0,True)
```

以上函数代码很简单，与 mx_funlst 批量调用函数类似，读者可以对照进行解读。

13.5.6 案例 13-9：批量加载算法模型

做生意讲究的是有进有出，同样，我们储存算法模型的最终目的也是为了现实应用。

前面介绍了批量储存算法模型的案例，下面介绍批量读取（加载）储存算法模型的方法。

原以为批量读取储存算法模型比批量加载储存算法模型简单，经过实际案例测试才发现，批量读取比批量储存更复杂。

其原因在于，储存只要把数据保存到磁盘就行了，而批量读取数据时，读取后的数据会直接计入程序的数据处理流程中，如果处理不当，不但不会提升整体性能，反而会干扰其他数据，造成额外错误。

案例 13-9 的文件名是 zai409_mx_06.py，介绍批量加载保存的机器学习算法模型并进行测试。下面分组对代码进行讲解。

第 1 组代码，设置参数，读取测试数据：

```
#1
fsr0='dat/ccpp_'
```

```
print('#1',fsr0)
mxlst=['line','log','bayes','knn','forest','dtree','svm','mlp','mlpre
g']
x_test=zai.ai_f_datRd010(fsr0+'xtest.csv')
y_test=zai.ai_f_datRd010(fsr0+'ytest.csv',1)
```

因为有存储的算法模型，所以无须读取训练数据，只需要读取测试数据和对应的结果数据即可，。这里没有使用 `ai_dat_rd` 多组数据读取函数，调用的是 `zai.ai_f_datRd010` 单组数据读取函数，函数代码如下：

```
def ai_f_datRd010(fsr,k0=0,fgPr=False):
    #1
    df=pd.read_csv(fsr,index_col=False);
    #2
    if k0>0:
        ysgn=df.columns[0];#print('y',ysgn)
        df[ysgn]=round(df[ysgn]*k0).astype(int)

    #3
    if fgPr:
        print('\n',fsr);print(df.tail())

    #4
    return df
```

在 `ai_f_datRd010` 单组数据读取函数中，如果读取主数据，则参数 `k0=0`；如果读取以 `y` 开头的答案数据，则需要进行缩放，并且转换为整数格式。

在第 1 组代码中，`x_test` 主数据和结果数据 `y_test` 的调用参数是不同的。

此外，为了保持函数格式的一致性，我们把 `ai_dat_rd` 多组数据读取函数统一归纳在 `ai_f_xxx` 文件函数中，将函数名称改为：

```
def ai_f_datRd(fsr0,k0=1,fgPr=False):
```

为了兼容以往的案例，原 `ai_dat_rd` 函数依然保留，在以后的版本中进行优化处理。

第 2 组代码，批量读取算法模型：

```
#2
zai.xmodel={}
print('\n#2,ai_f_mxRdlst')
zai.ai_f_mxRdlst(fsr0,mxlst)
```


调用的是 `zai.ai_f_mxRdlst` 函数批量读取各个算法模型文件，对应的函数代码如下：

```
def ai_f_mxRdlst(fsr0,funlst):
    for funSgn in funlst:
        fss=fsr0+funSgn+'.pkl'
        print(fss)
        xmodel[funSgn]=joblib.load(fss)
```

读取其实很简单，难点在于读取后如何处理数据。

为了提高性能，我们在 `ztop_ai` 极宽机器学习模块中，设置了一个字典格式的全局变量 `zai.xmodel`，并且在批量调用前进行清零处理。

第 3 组代码，批量调用算法模型，并进行分析预测：

```
#3
print('\n#3,mx_funlst8mx')
zai.mx_funlst8mx(mxlst, x_test, y_test,yk0=5,fgInt=False)
```

实际分析预测调用的是函数 `zai.mx_funlst8mx`，对应的代码如下：

```
def mx_funlst8mx(mxlst, x_test, y_test,yk0=5,fgInt=False):
    for msgn in mxlst:
        print('@msgn:',msgn)
        tim0=arrow.now()
        mx=xmodel[msgn]
        mx_fun8mx(mx, x_test, y_test,yk0,fgInt)
        zt.timNSec('',tim0,True)
```

`zai.mx_funlst8mx` 函数代码很简单，关键就在于：

```
mx=xmodel[msgn]
```

使用 `xmodel` 中保存的模型并进行分析预测。

变量 `yk0` 是误差精度的 K 阈值，通常，我们使用的都是 `yk0=5`，即结果误差在 5% 以内，下面把第 3 组调用代码的参数设置为 `yk0=1`，按 1% 的误差精度再进行一次测试。

第 3 组代码分析的结果是：

#3,mx_funlst8mx	#3,mx_funlst8mx
@msgn: line	@msgn: line
@mx:mx_sum,kok:99.96%	@mx:mx_sum,kok:68.52%
0.01 s, 13:10:01 ,t0, 13:10:01	0.01 s, 13:16:34 ,t0, 13:16:34
@msgn: log	@msgn: log

@mx:mx_sum,kok:99.67%	@mx:mx_sum,kok:53.64%
0.01 s, 13:10:01 ,t0, 13:10:01	0.01 s, 13:16:34 ,t0, 13:16:34
@msgn: bayes	@msgn: bayes
@mx:mx_sum,kok:99.83%	@mx:mx_sum,kok:54.72%
0.01 s, 13:10:01 ,t0, 13:10:01	0.01 s, 13:16:34 ,t0, 13:16:34
@msgn: knn	@msgn: knn
@mx:mx_sum,kok:99.50%	@mx:mx_sum,kok:66.14%
0.03 s, 13:10:01 ,t0, 13:10:01	0.03 s, 13:16:34 ,t0, 13:16:34
@msgn: forest	@msgn: forest
@mx:mx_sum,kok:99.67%	@mx:mx_sum,kok:78.64%
0.02 s, 13:10:01 ,t0, 13:10:01	0.02 s, 13:16:34 ,t0, 13:16:34
@msgn: dtree	@msgn: dtree
@mx:mx_sum,kok:99.75%	@mx:mx_sum,kok:75.59%
0.01 s, 13:10:01 ,t0, 13:10:01	0.01 s, 13:16:34 ,t0, 13:16:34
@msgn: svm	@msgn: svm
@mx:mx_sum,kok:96.40%	@mx:mx_sum,kok:71.53%
1.84 s, 13:10:03 ,t0, 13:10:01	1.83 s, 13:16:36 ,t0, 13:16:34
@msgn: mlp	@msgn: mlp
@mx:mx_sum,kok:99.83%	@mx:mx_sum,kok:47.95%
0.02 s, 13:10:03 ,t0, 13:10:03	0.01 s, 13:16:36 ,t0, 13:16:36
@msgn: mlpreg	@msgn: mlpreg
@mx:mx_sum,kok:99.96%	@mx:mx_sum,kok:62.37%
0.01s, 13:10:03 ,t0, 13:10:03	0.01 s, 13:16:36 ,t0, 13:16:36
5%误差精度, yk0=5	1%误差精度, yk0=1

将以上结果进行对比可以看出：

- 除了 SVM 向量机算法运行速度是 1.8 秒以外，其他机器学习算法的运行速度都在零点零几秒左右，而且运行速度与 yk0 误差精度没有关系。
- 当 yk0=5、误差精度为 5%时，除 SVM 向量机算法准确度是 96%以外，其他算法的准确度都在 99%左右。
- 当 yk0=1、误差精度为 1%时，准确度最高的是随机森林算法的 78.64%，最低的是 MLP 神经网络算法的 47.95%。
- 有趣的是，当误差精度为 1%时，原本效果最差的 SVM 向量机算法准确度为 71.5%，名列前茅。

13.6 机器学习组合算法

本章前面介绍了不少案例，其中大部分是为了给机器学习组合算法做铺垫。

机器学习组合算法看起来简单，其实有以下几个难点。

- 如何集成多种机器学习算法模型并进行测试。
- 如何评估多个算法模型的预测结果，并得出最佳结果数据。
- 采用什么样的机器学习算法组合最好。

针对以上难点，实盘往往会采用权重的模式，这样大大增加了算法和程序的复杂程度，对于实盘效果的提升也非常有限，因此采用统一权重，所有参数的权重都为 1。

在实盘测试中发现，并非组合的算法模型越多，最终的结果准确度越高，有时，模型种类多的算法组合的准确度，低于种类少的算法组合的准确度。

这一点，我们在本章的案例中也会介绍，这也从机器学习组合算法的角度证明了笔者小数据算法的合理性。

13.6.1 案例 13-10：机器学习组合算法

机器学习组合算法，也可以称为机器学习集成算法。

案例 13-10 的文件名是 `zai410_xsum.py`，介绍使用机器学习集成算法，以及对各种机器学习组合算法进行性能测试。下面分组介绍相关的程序代码。

第 1 组代码，设置相关的参数，并读取测试所需的数据集：

```
#1
fsr0='dat/ccpp_'
print('#1',fsr0)
mlst1=['line','log','bayes','knn','forest','dtree','svm','mlp','mlpre
g']
mlst2=['forest','dtree','svm']
mlst3=['line','log','bayes']
mlst4=['line','forest','dtree']

x_test=zai.ai_f_datRd010(fsr0+'xtest.csv')
y_test=zai.ai_f_datRd010(fsr0+'ytest.csv',1)
```

在以上代码中，需要注意的是 `mlst` 多组列表变量，代表了不同的机器学习算法组合。

第 2 组代码，从文件中批量读取机器学习算法模型，并保存到全局变量 `zai.xmodel`：

```
#2
zai.xmodel={}
print('\n#2,ai_f_mxRdlst')
zai.ai_f_mxRdlst(fsr0,mlst1)
```

批量读取输入参数，`mlst1` 必须覆盖程序中所需的各种机器学习算法标识符，但是没有现成的列表变量，需要采用集合运算生成一个新的列表数据合集。

第 3 组代码，使用 `mlst1` 列表参数调用 `zai.mx_mul` 机器学习组合算法函数：

```
#3
print('\n#3,mlst1',mlst1)
zai.mx_mul(mlst1, x_test, y_test,yk0=1,fgInt=False,fgDebug= False)
```

由于第 3 组至第 6 组代码基本相同，只是调用参数不同，所以合并相关的输出信息，一起解读。

第 3 组至第 6 组代码，对应的输出数据分别如下：

```
#3,mlst1 ['line', 'log', 'bayes',      #4,mlst2 ['forest', 'dtree',
'knn', 'forest', 'dtree', 'svm', 'mlp','svm']
'mlpreg']

y_pred,预测                                y_pred,预测
1 y_pred01,kok:68.52% line 0.0 s s          1 y_pred01,kok:78.64% forest 0.02
2 y_pred02,kok:53.64% log 0.0 s             2 y_pred02,kok:75.59% dtree 0.0 s
3 y_pred03,kok:54.72% bayes 0.0 s           3 y_pred03,kok:71.53% svm 1.9 s
4 y_pred04,kok:66.14% knn 0.03 s             @mx:mx_sum,kok:78.60%
5 y_pred05,kok:78.64% forest 0.02 s         第 4 组输出信息
6 y_pred06,kok:75.59% dtree 0.0 s
7 y_pred07,kok:71.53% svm 1.92 s
8 y_pred08,kok:47.95% mlp 0.01 s
9 y_pred09,kok:62.37% mlpreg 0.0 s
@mx:mx_sum,kok:74.37%

第 3 组输出信息
#5,mlst3 ['line', 'log', 'bayes']          #6,mlst3 ['line', 'forest', 'dtree']
```

y_pred, 预测

1 y_pred01,kok:68.52% line 0.0 s

2 y_pred02,kok:53.64% log 0.0 s

3 y_pred03,kok:54.72% bayes 0.0 s

@mx:mx_sum,kok:64.34%

第 5 组输出信息

y_pred, 预测

1 y_pred01,kok:68.52% line 0.0 s

2 y_pred02,kok:78.64% forest 0.02

3 y_pred03,kok:75.59% dtree 0.0 s

@mx:mx_sum,kok:81.98%

第 6 组输出信息

在以上输出信息中，mx_sum 代表的是机器学习组合算法的综合结果。

由以上结果可以看出：

- 准确度最高的是第 6 组 81.98%，准确度最低的是第 5 组 64.34%；
- 除了第 6 组以外，其他组合算法的最终结果都低于组合中的最高单个算法数值。

以上案例结果，从机器学习组合算法的角度证明了笔者小数据算法的合理性。

13.6.2 机器学习组合算法函数

zai.mx_mul 机器学习组合算法函数位于 ztop_ai 极宽机器学习模块中，函数代码如下：

```
def mx_mul(mlst, x_test, y_test, yk0=5, fgInt=False, fgDebug=False):
    #1
    print('\ny_pred, 预测')
    df9, xc, mxn9 = x_test.copy(), 0, len(mlst)
    df9['y_test'] = y_test
    #2
    for msgn in mlst:
        xc += 1; tim0 = arrow.now()
        mx = xmodel[msgn]
        y_pred = mx.predict(x_test.values)
        #3
        if xc == 1: df9['y_sum'] = y_pred
        else: df9['y_sum'] = df9['y_sum'] + y_pred
```

```

#4
tn=zt.timNSec('',tim0)
df9['y_pred']=y_pred
#5
dacc=ai_acc_xed(df9,1,fgDebug)
xss='y_pred{0:02},kok:{1:.2f}%'.format(xc,dacc);print
(xc,xss,msgn,tn,'s')
    ysgn='y_pred'+str(xc);df9[ysgn]=y_pred
#6
df9['y_pred']=df9['y_sum']/mxn9
dacc=ai_acc_xed(df9,yk0,fgDebug)

#7
if fgDebug:
    df9.to_csv('tmp/df9_pred.csv');

#8
print('@mx:mx_sum,kok:{0:.2f}%'.format(dacc))
return dacc,df9

```

zai.mx_mul 机器学习组合算法函数的代码并不复杂，重点是第 2 组代码：

```
for msgn in mlst:
```

通过迭代循环，计算各个机器学习算法的结果与准确度，并计算累加的总的准确度，直到第 6 组代码，对总的准确度进行平均计算。

其他的代码前面已经讲述过，此处不再赘述。

14

第 14 章

足彩机器学习模型构建

前面几章讲解了多种常用的机器学习算法，本章将结合足彩实盘数据和各种机器学习算法，采用最新的人工智能技术，通过足彩数据进行专业的分析，构建各种机器学习的算法模型，再采用这些模型进行实盘的结果预测。

14.1 数据整理

俗话说“兵马未动，粮草先行”。对于机器学习、数据分析而言，这里面的“粮草”指的就是各种数据。

前面已经学习过 `gid` 比赛数据和 `xdat` 赔率数据的下载和转换，但前面的案例偏重于让读者理解，而不是实盘。

从本章开始，案例会尽量偏重实盘分析，采用实盘的标准来要求各个环节。

`gid` 比赛数据只有一个文件 `/tbfdDat/gid2017.dat`，相对比较简单，无需额外处理。

需要处理的是 `/tbfdDat/xdat/` 目录下的赔率数据文件，尽管已经进行了初步处理，按 `gid` 比赛编码分割，完善 `gid` 比赛信息等，但还是需要进行进一步的数据整理。

- 把分散的赔率数据文件合并成一个独立的赔率数据文件 `/tbfdDat/xdat2017.dat`。
- 合并时，需要进行基本的数据清洗工作，删除其中 `kwin=-1` 即没有完成的比赛或者半途中止的比赛的数据。
- 按年度采用独立和累加两种模式，分割赔率数据文件 `xdat2017.dat`，并保存到

/tbfDat/mdat/数据目录。

这些清洗、分割后的数据文件就是我们进行机器学习、构建各种算法模型的数据源，在稍后的案例中会具体介绍。

严格来说，这些赔率数据只是最原始的数据源，属于源数据，通常还应该进行数据归一化处理，这方面 **Sklearn** 中有多组函数。

不过，因为本书属于入门性质，同时限于篇幅，并没有介绍 **Sklearn** 的数据整理模块，这部分内容读者可以自己学习。

我们没有对赔率数据进行二次处理的另外一个原因是，赔率数据本身就是非常规范的数据源，类似金融股市中的数据，全球采用的标准都是相同的，特别是欧赔数据，各个博彩机构之间的赔率数据可以直接进行对比分析。

所以，笔者常说，足彩数据是最好的大数据分析数据源，也是最好的人工智能、机器学习的数据源。

14.1.1 案例 14-1：赔率数据合成

案例 14-1 的文件名是 `zd101_dat_xlnk.py`，用于合并 `/tbfDat/xdat/` 目录下各个独立的赔率数据文件。

- 文件名中的 `lnk` 是英文 `link` 的缩写。
- 文件名中的字符 `x`，表示在连接前需要进行数据整理工作。

这里的数据整理工作，主要是删除其中 `kwin=-1` 即没有完成的比赛或者半途中止的比赛的数据。

案例程序很简单，核心代码如下：

```
rs0='/tbfDat/'
rxdat=rs0+'xdat/'
ftg='tmp/xdat2017.dat'

#-----
tft.fb_xdat_xlnk(rxdat,ftg)
```

以下程序的主要工作是设置数据目录和合并后的数据文件名，以及调用 `tft_tools` 模块中的赔率数据连接函数 `tft.fb_xdat_xlnk`，该函数代码如下：

```
def fb_xdat_xlnk(rs0,ftg):
```



```

flst=zt.lst4dir(rs0)
df9=pd.DataFrame(columns=tfsys.gxdatSgn,dtype=str)
for xc,fs0 in enumerate(flst):
    fss=rs0+fs0
    print(xc,fss)
    df=pd.read_csv(fss,index_col=False,dtype=str,encoding=
'gb18030')
    #
    df2=df[df['kwin']!='-1']
    df9=df9.append(df2,ignore_index=True)
    #
    if (xc % 2000)==0:
        #df9.to_csv(ftg,index=False,encoding='gb18030')
        fs2='tmp/x_'+str(xc)+'.dat';print(fs2,fss)
        df9.to_csv(fs2,index=False,encoding='gb18030')
    #
    df9.to_csv(ftg,index=False,encoding='gb18030')

```

tft.fb_xdat_xlnk 赔率数据连接函数位于 tfb_tools 模块中，下面按流程对代码进行简单的讲解。

- 调用 zt.lst4dir 函数，把目录下的文件名转换为列表数据并保存到变量 flst。
- 定义 df9 结果变量，采用 Pandas 的 DataFrame 格式。
- 对 flst 文件名列表采用迭代循环，注意 enumerate 函数。
 - 读取单个的赔率数据文件到变量 df。
 - 删除 kwin=-1 的无效比赛数据，并追加到 df9 结果变量中。
 - 如果计数器 xc 是 2000 的整数，则取模运算采用百分符号%，表示每处理 2000 个文件，保存一次中间数据，注意文件名称包含了 xc 计数器。
 - 文件格式使用 GB18030，因为 500 彩票网站的数据源使用 GBK 编码，赔率数据中有中文字符串，采用 GBK 编码偶尔会出现错误字符，所以选择使用 GB18030 编码格式。
- 完成所有循环迭代，把结果数据保存到数据文件 tmp/xdat2017.dat。

最后，需要手动把最终赔率数据文件 tmp/xdat2017.dat 移动到 tfbDat 目录下。原本程序可以自动保存在 tfbDat 目录下，也是很简单的代码，不过对于这种需要长时间运算的程序，笔者建议最后的结果还是使用手动模式进行处理，以免出现各种意外错误，毁坏原来的数据文件。

这个也是笔者在量化交易中一直坚持以策略为核心、人工下单，而不是片面追求程序化交易、自动下单的原因之一。

案例 14-1 的程序代码虽然很简单，但运行非常耗时，笔者的 i7 笔记本、SSD 固态硬盘运行了将近 15 个小时，如图 14-1 所示。

E:\zwPython\zc_demo\tmp							
名称	大小	修改日期	类型				
x_4000.dat	22,286 KB	2017/2/19 20:18	DAT 文件	x_36000.dat	205,682 KB	2017/2/20 0:23	DAT 文件
x_6000.dat	33,759 KB	2017/2/19 20:22	DAT 文件	x_38000.dat	217,182 KB	2017/2/20 0:51	DAT 文件
x_8000.dat	45,229 KB	2017/2/19 20:27	DAT 文件	x_40000.dat	228,731 KB	2017/2/20 1:21	DAT 文件
x_10000.dat	56,656 KB	2017/2/19 20:34	DAT 文件	x_42000.dat	240,322 KB	2017/2/20 1:52	DAT 文件
x_12000.dat	68,046 KB	2017/2/19 20:43	DAT 文件	x_44000.dat	251,754 KB	2017/2/20 2:25	DAT 文件
x_14000.dat	79,530 KB	2017/2/19 20:53	DAT 文件	x_46000.dat	263,220 KB	2017/2/20 2:59	DAT 文件
x_16000.dat	90,966 KB	2017/2/19 21:04	DAT 文件	x_48000.dat	274,642 KB	2017/2/20 3:35	DAT 文件
x_18000.dat	102,394 KB	2017/2/19 21:17	DAT 文件	x_50000.dat	286,072 KB	2017/2/20 4:13	DAT 文件
x_20000.dat	113,750 KB	2017/2/19 21:32	DAT 文件	x_52000.dat	297,707 KB	2017/2/20 4:52	DAT 文件
x_22000.dat	125,231 KB	2017/2/19 21:48	DAT 文件	x_54000.dat	309,261 KB	2017/2/20 5:32	DAT 文件
x_24000.dat	136,715 KB	2017/2/19 22:05	DAT 文件	x_56000.dat	320,737 KB	2017/2/20 6:14	DAT 文件
x_26000.dat	148,133 KB	2017/2/19 22:25	DAT 文件	x_58000.dat	332,278 KB	2017/2/20 6:58	DAT 文件
x_28000.dat	159,597 KB	2017/2/19 22:45	DAT 文件	x_60000.dat	343,896 KB	2017/2/20 7:43	DAT 文件
x_30000.dat	171,104 KB	2017/2/19 23:07	DAT 文件	x_62000.dat	355,519 KB	2017/2/20 8:30	DAT 文件
x_32000.dat	182,663 KB	2017/2/19 23:31	DAT 文件	x_64000.dat	367,104 KB	2017/2/20 9:18	DAT 文件
x_34000.dat	194,190 KB	2017/2/19 23:56	DAT 文件	x_66000.dat	378,098 KB	2017/2/20 10:09	DAT 文件
				x_68000.dat	389,271 KB	2017/2/20 11:02	DAT 文件
				xdat2017.dat	400,463 KB	2017/2/20 11:56	DAT 文件

图 14-1 赔率数据临时合并文件

从图 14-1 中可以看出：

- 最早的 x_0.dat 文件被删除了，第一个有记录的数据文件是 x_4000.dat；
- x_4000.dat 文件的时间记录是 2 月 19 日 20 点 18 分；
- 每 2000 个赔率文件合并后占据的空间约 10MB，这个数值从程序开始到最后都比较稳定；
- 程序开始运行时，处理 2000 个文件只需 4~5 分钟，到最后快结束时处理 2000 个文件需要将近 1 个小时，这是因为结果变量 df9 的规模越来越大了；
- 合并后的赔率数据文件大约 400MB，共有 228 万条赔率数据；
- 程序运行时，会生成数十个中间数据文件，大约需要 10GB 空间，运行前请注意；
- 运行后请注意删除 tmp 临时文件目录，以释放硬盘空间；
- /tfbDat/xdат/数据源和程序目录最好都使用 SSD，以提高运行效率。

历年赔率数据的合并虽然耗时，但基本上是一次性的工作，通常每个比赛年度结束后进行一次合并即可。

14.1.2 案例 14-2：按年切割赔率数据

案例 14-2 用于切割合并后的赔率数据文件：

```
/tbfDat/xdat2017.dat/
```

- 按年度进行切割，自 2010 年起，每年创建一个以 year 开头的独立文件。
- 文件保存在 tmp 目录下，需要手动移动到目录/tbfDat/mdat/。
- 时间切割是按数据字段 tplay 进行切割的。

案例 15-2 的文件名是 zd102_dat_xcut01.py，源程序文件名是 zc402_dat_cutx.py，进行了部分修改。

核心代码如下：

```
rs0='/tbfDat/'
fdat=rs0+'xdat2017.dat'
xdat=pd.read_csv(fdat,index_col=False, dtype=str, encoding='gb18030')
#
ksgn='tplay'
ylst=['2010','2011','2012','2013','2014','2015','2016','2017']
#
ftg0='tmp/year_'
zdat.df_kcut8yearlst(xdat,ksgn,ftg0,ylst)
```

运行流程如下：

- 读取赔率数据文件，保存到变量 xdat；
- 设置参数；
- 调用 zdat.df_kcut8yearlst 年度数据切割函数进行数据切割。

生成的数据文件保存在 tmp 目录下，需要手动移动到目录/tbfDat/mdat/中。

案例 14-2 运行速度很快，2010—2017 年的年度数据全部切割完成不过 1 分钟的时间。

zdat.df_kcut8yearlst 年度数据切割函数在前面介绍过，这里不再重复介绍。

14.1.3 案例 14-3：累计切割赔率数据

案例 14-3 用于切割合并后的赔率数据文件：

```
/tbfDat/xdat2017.dat/
```

- 按年度进行合并切割，自 2011 年起，每年创建一个以 xsum 开头的独立文件，保存 2010 年至当年的累计赔率数据。
- 文件保存在 tmp 目录下，需要手动移动到目录/tbfDat/mdat/中。
- 时间切割是按数据字段 tplay 进行切割的。

案例 14-3 的文件名是 zd103_dat_xcut02.py，核心代码如下：

```
rs0='/tbfDat/'
fdat=rs0+'xdat2017.dat'
xdat=pd.read_csv(fdat,index_col=False,dtype=str,encoding='gb18030')
#
ksgn='tplay'
ylst=['2011','2012','2013','2014','2015','2016']
#
tim0str='2010-01-01'
ftg0='tmp/xsum_'
zdat.df_kcut8myearlst(xdat,ksgn,tim0str,ftg0,ylst)
```

案例 14-3 与案例 14-2 的代码基本相同，只是个别参数设置有所差异，此外，最大的差异在于最后一行代码：

```
zdat.df_kcut8myearlst(xdat,ksgn,tim0str,ftg0,ylst)
```

案例 14-3 调用的是累计年度切割函数 zdat.df_kcut8myearlst，函数代码如下：

```
def df_kcut8myearlst(df,ksgn,tim0str,ftg0,yearlst):
    for ystr in yearlst:
        tim9str=ystr+'-12-31'
        df2=df_kcut8tim(df,ksgn,tim0str,tim9str)
        ftg=ftg0+ystr+'.dat';print(ftg)
        df2.to_csv(ftg,index=False,encoding='gb18030')
```

以上函数代码与年度数据切割函数 zdat.df_kcut8yearlst 类似，只是起始时间是函数调用参数 tim0Str 设置的，不是每年的 1 月 1 日，案例中是 2010-01-01。

14.2 年度足彩赔率模型

年度足彩赔率模型是根据足彩赔率历史数据，按每年的赔率数据独立进行建模，相当于每年每种算法都是一种模型。

例如，使用 2016 年的赔率数据作为数据源，结合 Line 线性回归算法模型，就

是 2016 年度的线性回归算法模型，如果采用 KNN（K 近邻）算法，最终的结果模型就是 2016 年度的 KNN（K 近邻）算法模型。

14.2.1 案例 14-4：2016 年度足彩赔率模型组

案例 14-4 介绍通过 2016 年度赔率数据，结合各种算法模型，构建各种具体的足彩赔率算法模型，因为是多种算法，所以按年度数据将其称为 2016 足彩赔率算法模型组。

案例 14-4 是最简单的足彩模型构建案例，其实还可以更简单，就是单独构建一种算法的赔率模型，读者可以根据本节案例，自己编写相关的程序代码。

案例 14-4 跳过了单个算法模型案例，直接调用批量构建算法模型函数，这个也是实盘中的常用模式。

案例 14-4 的文件名是 zd104_mlib01.py，下面分组介绍相关代码。

第 1 组代码，设置数据源等输入的相关参数：

```
#1
rs0,ysgn='/tfbDat/','kwin'
fsr=rs0+'mdat/year_2016.dat'
mxlst=['line','log','bayes','knn','forest','gbdt','dtree','mlp','mlpre
g']
print('#1',fsr)
```

数据源文件使用文件/tfbDat/ mdat/year_2016.dat，表示 2016 年度的赔率数据。

mxlst 是算法名称列表，省略了 SVM 向量机算法，因为太耗时间。

第 2 组代码，设置模型相关参数：

```
#
#2
ftg0='tmp/p3y2016_'
xlst=['pwin0','pdraw0','plost0']
```

其中，xlst 是对应的主数据集的字段名列表，因为这里只使用了 3 个最核心的字段，即开盘胜、开盘平、开盘负赔率，所以结果文件名以 p3 开头，y 是英文 year 的缩写，2016 表示年度数据。再加上机器学习算法名称，就构成了完整的足彩算法模型文件名。例如，使用 line 线性回归算法，2016 年度三字段的足彩赔率数据文件名是 p3y2016_line.pkl，文件后缀名是.pkl。

第 3 组代码, 读取训练数据集, 主数据 `x_dat` 和对应的答案数据 `y_dat` 代码如下:

```
#
#3
print('\n#3,x_dat,y_dat')
x_dat,y_dat= tft.fb_xdat_xrd020(fsr,xlst,ysgn,1,True)
```

读取数据使用的是一个全新的 `tft.fb_xdat_xrd020` 数据读取函数, 相关代码如下:

```
def fb_xdat_xrd020(fsr,xlst,ysgn='kwin',k0=1,fgPr=False):

    #1
    df=pd.read_csv(fsr,index_col=False,encoding='gb18030')
    #2
    if ysgn=='kwin':
        df[ysgn]=df[ysgn].astype(str)
        df[ysgn].replace('3','2', inplace=True)
    #3
    df[ysgn]=df[ysgn].astype(float)
    df[ysgn]=round(df[ysgn]*k0).astype(int)
    #4
    x_dat,y_dat= df[xlst],df[ysgn]

    #5
    if fgPr:
        print('\n',fsr);
        print('\nx_dat');print(x_dat.tail())
        print('\ny_dat');print(y_dat.tail())
        #df.to_csv('tmp\df.csv',index=False,encoding='gb18030')
    #6
    return x_dat,y_dat
```

函数代码与之前介绍的 `zai.ai_f_datRd010` 数据读取函数类似, 需要注意的是其中第 2 组的代码:

```
#2
if ysgn=='kwin':
    df[ysgn]=df[ysgn].astype(str)
    df[ysgn].replace('3','2', inplace=True)
```

需要把比赛结果中的数字 3 改为 2, 因为胜、平、负的数字代码分别是 3、1、0, 而 3 和 1 之间缺一个 2, 在实盘分析时很容易引起误判, 所以把胜盘的代码由数字 3

改为 2。

因为是足彩专用的算法，所以函数也有通用机器学习模块库，将其改为 `tfb_tools` 足彩工具函数模块库。

其他代码没有额外的难点，读者自己解读即可。

第 4 组代码，调用 `zai.ai_f_mxWrlst` 批量构建算法模型函数，批量生成各种相关的足彩赔率模型，并保存到相关的文件：

```
#4
print('\n#4,ai_f_mxWrlst')
zai.ai_f_mxWrlst(ftg0,mx1st,x_dat,y_dat)
```

第 4 组程序，对应的输出信息是：

```
4,ai_f_mxWrlst

tmp/p3y2016_line.pkl
0.11 s, 15:09:37 ,t0, 15:09:37

tmp/p3y2016_log.pkl
1.62 s, 15:09:39 ,t0, 15:09:37

tmp/p3y2016_bayes.pkl
0.07 s, 15:09:39 ,t0, 15:09:39

tmp/p3y2016_knn.pkl
1.11 s, 15:09:40 ,t0, 15:09:39

tmp/p3y2016_forest.pkl
3.32 s, 15:09:43 ,t0, 15:09:40

tmp/p3y2016_gbdtd.pkl
128.48 s, 15:11:52 ,t0, 15:09:43

tmp/p3y2016_dtree.pkl
0.87 s, 15:11:53 ,t0, 15:11:52

tmp/p3y2016_mlp.pkl
25.19 s, 15:12:18 ,t0, 15:11:53
```

```
tmp/p3y2016_mlpreg.pkl
16.79 s, 15:12:35 ,t0, 15:12:18
```

如图 14-2 所示是以上各种算法模型的运行时间对比。

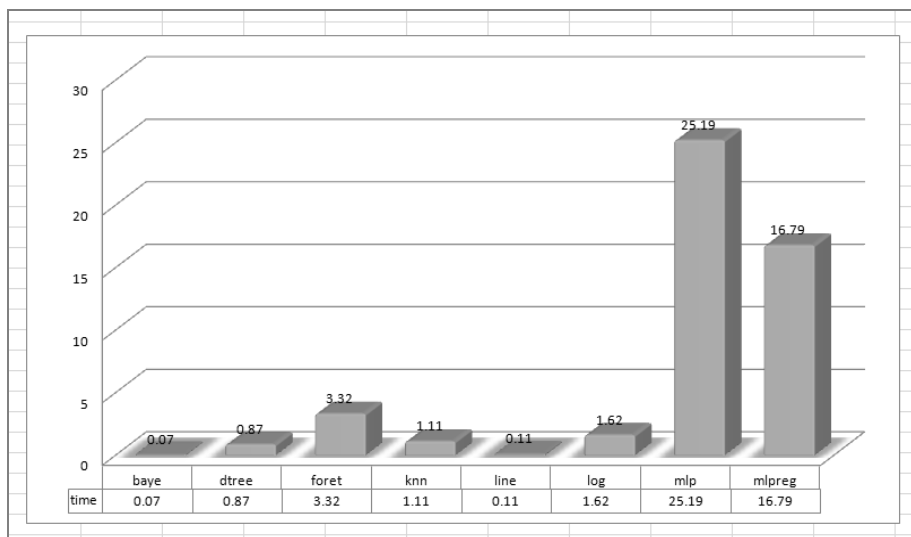


图 14-2 各种算法模型的运行时间对比

由以上输出信息和图 14-2 可以看出：

- 贝叶斯算法模型构建速度最快，才 0.07 秒；
- GBDT 迭代决策树算法耗时最长，需要 128 秒，比其他各种算法的总时间还长，删除的 SVM 向量机算法的耗时比 GBDT 算法耗时还要长，所以删除了，有兴趣的读者可以自己先用一个月的赔率数据测试运行，构建 SVM 模型的时间。图 14-2 中省略了 GBDT 迭代决策树算法模型的数据；
- MLP 和 MLPreg 神经网络算法的模型构建时间差不多，分别是 25 秒和 16.8 秒；
- 生成的足彩模型文件保存在 tmp 目录下，需要人工复制到目录/tfbDat/mlib/中。

14.2.2 案例 14-5：年度多字段足彩赔率模型组

前面的案例 14-4 使用的是 2016 年度的赔率数据，结合各种算法模型，构建具体的足彩赔率算法模型。

案例 14-4 的第 2 组代码是设置模型相关参数：

```
#
#2
ftg0='tmp/p3y2016_'
xlst=['pwin0','pdraw0','plost0']
```

其中，`xlst` 变量只使用 3 个最核心的字段，即开盘胜、平、负的赔率，所以结果文件名以 `p3` 开头。

案例 14-5 的文件名是 `zd105_mlib02.py`，介绍多字段足彩赔率模型组，除了第 2 组代码，其他代码与案例 14-4 完全一样。

案例 14-5 的第 2 组代码如下：

```
#
#2.a
#ftg0='tmp/p3y2016_'
#xlst=['pwin0','pdraw0','plost0']

#2.b
ftg0='tmp/p7y2016_'
xlst=['cid','pwin0','pdraw0','plost0','pwin9','pdraw9','plost9']
'''

#2.c
ftg0='tmp/pzy2016_'
xlst=['cid','pwin0','pdraw0','plost0','pwin9','pdraw9','plost9'
      , 'vwin0','vdraw0','vlost0','vwin9','vdraw9','vlost9','vback0','v'
back9'
      , 'vwin0kali','vdraw0kali','vlost0kali','vwin9kali','vdraw9kali',
'vlost9kali'
      , 'mtid','gtid','tweek']
'''
```

与以往的案例不同，第 2 组代码又分为 2.a、2.b 和 2.c 三段，每次运行只能使用其中一段代码，需要屏蔽其他两段代码。

2.a 段代码其实就是案例 14-4 中已经使用过的 3 字段参数设置。

2.b 段代码用于 7 字段参数设置，输出的赔率模型文件名开头是 `p7y2016_`。

`xlst` 使用的数据字段名包括：博彩机构代码，开盘胜、平、负赔率，收盘胜、平、负赔率等数据。

2.c 段代码用于全字段参数设置，输出的赔率模型文件名开头是 `pzy2016_`。

所谓全字段参数，就是在 `xlst` 列表中使用赔率数据中的所有字段来构建算法模型，不过 `Sklearn` 的算法模型函数只支持数值，所以删除了部分字符串格式的字段。

案例 14-5 有一个克隆版本的程序，可以称做案例 14-5z，案例 14-5z 程序文件名是 `zd105_mlib02z.py`，就是在案例 14-5 的文件名后加了个字母 `z`。

在案例 14-5 程序中，`xlst` 使用的是 7 字段数据，而在案例 14-5z 程序中，`xlst` 使用的是全字段数据。

如图 14-3 所示为构建多字段足彩赔率模型的两个程序的输出信息对比。

<pre>#4,ai_f_mxWrlst tmp/p7y2016_line.pkl 0.1 s, 15:36:23 ,t0, 15:36:23 tmp/p7y2016_log.pkl 4.84 s, 15:36:28 ,t0, 15:36:23 tmp/p7y2016_bayes.pkl 0.08 s, 15:36:28 ,t0, 15:36:28 tmp/p7y2016_knn.pkl 3.83 s, 15:36:32 ,t0, 15:36:28 tmp/p7y2016_forest.pkl 6.06 s, 15:36:38 ,t0, 15:36:32 tmp/p7y2016_gbdtd.pkl 191.3 s, 15:39:49 ,t0, 15:36:38 tmp/p7y2016_dtree.pkl 2.42 s, 15:39:52 ,t0, 15:39:49 tmp/p7y2016_mlp.pkl 118.14 s, 15:41:50 ,t0, 15:39:52 tmp/p7y2016_mlpreg.pkl 7.69 s, 15:41:58 ,t0, 15:41:50 案例 14-5 构建 7 字段足彩赔率模型</pre>	<pre>#4,ai_f_mxWrlst tmp/pzy2016_line.pkl 0.48 s, 15:40:10 ,t0, 15:40:09 tmp/pzy2016_log.pkl 22.29 s, 15:40:32 ,t0, 15:40:10 tmp/pzy2016_bayes.pkl 0.15 s, 15:40:32 ,t0, 15:40:32 tmp/pzy2016_knn.pkl 4.06 s, 15:40:36 ,t0, 15:40:32 tmp/pzy2016_forest.pkl 11.99 s, 15:40:48 ,t0, 15:40:36 tmp/pzy2016_gbdtd.pkl 460.56 s, 15:48:29 ,t0, 15:40:48 tmp/pzy2016_dtree.pkl 8.67 s, 15:48:37 ,t0, 15:48:29 tmp/pzy2016_mlp.pkl 189.34 s, 15:51:47 ,t0, 15:48:37 tmp/pzy2016_mlpreg.pkl 8.39 s, 15:51:55 ,t0, 15:51:47 案例 14-5z 构建全字段足彩赔率模型</pre>
---	--

图 14-3 构建足彩赔率模型

如图 14-4 所示是构建多字段足彩赔率模型时各种机器学习算法的运算时间对比。

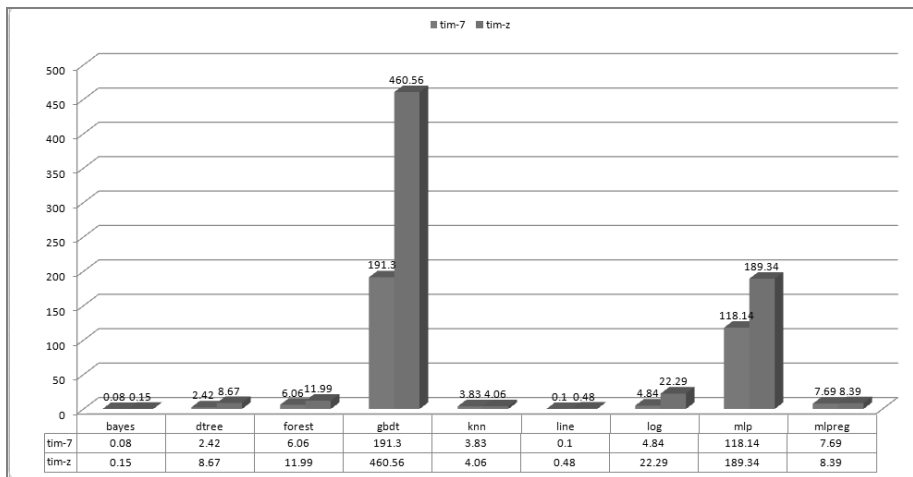


图 14-4 构建足彩赔率模型运算时间对比

在图 14-4 中，tim-7、tim-z 分别是构建 7 字段、全字段各种机器学习算法模型的程序运行所需的时间。

由图 14-3 和图 14-4 可以看出，构建全字段足彩赔率模型的时间几乎是构建 7 字段足彩赔率模型时间的数倍，基本上和字段数量成正比。

生成的足彩模型文件保存在 tmp 目录下，需要人工复制到目录/tfbDat/mlib/中。

本案例中只构建了 2016 年度的足彩赔率模型，读者可以参考以上案例，构建其他年度的足彩赔率模型。

14.3 累计足彩赔率模型

多年足彩赔率模型就是以 2010 年为起始年份，根据年度累计足彩赔率历史数据进行建模。

14.3.1 案例 14-6：累计 2016 足彩赔率模型组

案例 14-6 使用 2016 年度累计足彩赔率，结合各种算法模型，构建具体的足彩

赔率算法模型，因为是多种算法，所以按年度数据将其称为累计 2016 足彩赔率算法模型组。

案例 14-6 的文件名是 `zd106_msum01.py`，下面逐一介绍相关代码。

第 1 组代码，设置数据源等输入的相关参数：

```
#1
rs0,ysgn='/tfbDat/','kwin'
fsr=rs0+'mdat/xsumr_2016.dat'
mxlst=['line','log','bayes','knn','forest','gbdt','dtree','mlp','mlpr
eg']
print('#1',fsr)
```

数据源使用的是文件 `/tfbDat/ mdat/xsum_2016.dat`，表示 2016 年度的累计赔率数据。

`mxlst` 是算法名称列表，省略了 SVM 向量机算法，因为太耗时间。

第 2 组代码，设置模型相关参数：

```
#
#2
ftg0='tmp/m3y2016_'
xlst=['pwin0','pdraw0','plost0']
```

其中，`xlst` 是对应的主数据集的字段名列表，有 3 个最核心的字段，即开盘胜、平、负的赔率，因此结果文件名以 `m3` 开头。

第 3 组代码，读取训练数据集，即主数据 `x_dat` 和对应的答案数据 `y_dat`：

```
#
#3
print('\n#3,x_dat,y_dat')
x_dat,y_dat=tft.fb_xdat_xrd020(fsr,xlst,ysgn,1,True)
```

第 4 组代码，调用 `zai.ai_f_mxWrlst` 批量构建算法模型函数，批量生成各种相关的足彩赔率模型并保存到相关文件：

```
#4
print('\n#4,ai_f_mxWrlst')
zai.ai_f_mxWrlst(ftg0,mxlst,x_dat,y_dat)
```

由以上代码可以看出，案例 14-6 累计 2016 足彩赔率模型组与案例 14-4 年度 2016 足彩赔率模型组的代码基本相同，除了以下两个地方。

- 第 1 组代码中的数据源文件不同，案例 14-6 累计赔率模型使用 `xsum_2016.dat` 累计赔率数据文件，而案例 14-4 年度赔率模型使用 `year_2016.dat` 年度赔率数据

文件。

- 第 1 组代码中的输出文件名前缀不同, 案例 14-6 累计赔率模型使用字母 m 开头, 而案例 14-4 年度赔率模型使用字母 y 开头。

案例 14-6 主要输出信息如下:

```
#4,ai_f_mxWrlst

tmp/m3y2016_line.pkl
0.27 s, 16:02:54 ,t0, 16:02:53

tmp/m3y2016_log.pkl
31.38 s, 16:03:25 ,t0, 16:02:54

tmp/m3y2016_bayes.pkl
0.5 s, 16:03:26 ,t0, 16:03:25

tmp/m3y2016_knn.pkl
53.55 s, 16:04:19 ,t0, 16:03:26

tmp/m3y2016_forest.pkl
44.28 s, 16:05:03 ,t0, 16:04:19

tmp/m3y2016_gbdn.pkl
1403.69 s, 16:28:27 ,t0, 16:05:03

tmp/m3y2016_dtree.pkl
11.13 s, 16:28:38 ,t0, 16:28:27

tmp/m3y2016_mlp.pkl
85.71 s, 16:30:04 ,t0, 16:28:38

tmp/m3y2016_mlpreg.pkl
30.8 s, 16:30:35 ,t0, 16:30:04
```

如图 14-5 所示是构建累计足彩赔率模型时各种机器学习算法的运算时间对比。

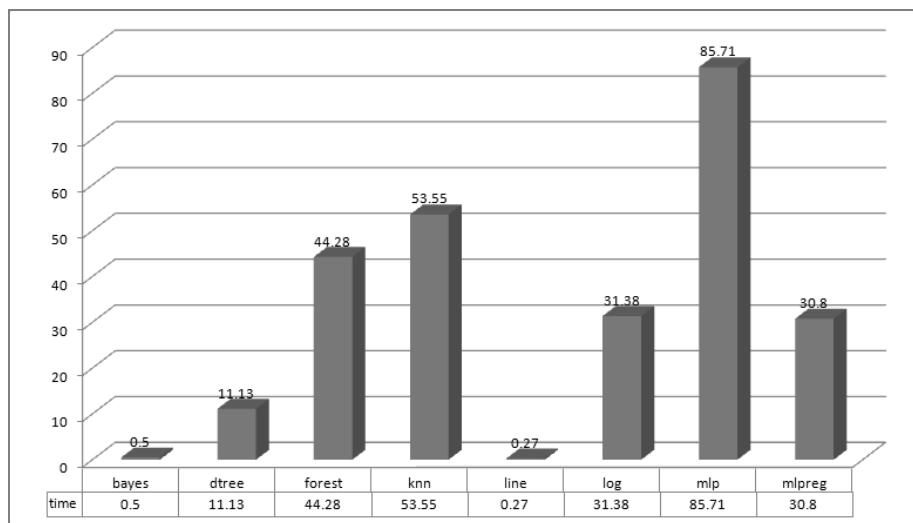


图 14-5 机器学习算法构建累计足彩赔率模型时间对比

由图 14-5 和案例输出信息可以看出：

- 线性回归算法模型构建速度最快，才 0.27 秒；
- GBDT 迭代决策树算法耗时最长，用 1403 秒约 24 分钟，比其他各种算法的总时间还长，图 14-5 中省略了 GBDT 的图形；
- MLP 神经网络算法和 MLPreg 神经网络算法的模型构建时间分别是 85 秒和 30 秒；
- 生成的足彩模型文件保存在 tmp 目录下，需要人工复制到目录/tfbDat/mlib/中。

14.3.2 案例 14-7：累计多字段足彩赔率模型组

前面的案例 14-6 使用 2016 年度的累计赔率数据，结合各种算法模型，构建具体的足彩累计赔率算法模型。

案例 14-6 的第 2 组代码设置模型的相关参数：

```
#
#2
ftg0='tmp/m3y2016_'
xlst=['pwin0','pdraw0','plost0']
```

其中，xlst 变量只使用 3 个最核心的字段，即开盘胜、平、负赔率，所以结果

文件名以 m3 开头。

案例 14-7 的文件名是 zd107_msum02.py, 介绍多字段累计足彩赔率模型组, 除了第 2 组代码, 其他代码与案例 14-6 完全一样。

案例 14-7 的第 2 组代码如下:

```
#2.a
#ftg0='tmp/m3y2016_'
#xlst=['pwin0','pdraw0','plost0']

#2.b
ftg0='tmp/m7y2016_'
xlst=['cid','pwin0','pdraw0','plost0','pwin9','pdraw9','plost9']
```

与案例 14-5 多字段足彩赔率模型组不同, 案例 14-7 的第 2 组代码只有 2.a、2.b 两段。

2.a 段代码是 3 字段参数设置, 输出的赔率模型文件名开头是 m3y2016_。

2.b 段代码是 7 字段参数设置, 输出的赔率模型文件名开头是 m7y2016_。

案例 14-7 主要输出信息如下:

```
#4,ai_f_mxWrlst

tmp/m7y2016_line.pkl
0.84 s, 16:03:06 ,t0, 16:03:05

tmp/m7y2016_log.pkl
37.26 s, 16:03:43 ,t0, 16:03:06

tmp/m7y2016_bayes.pkl
0.56 s, 16:03:44 ,t0, 16:03:43

tmp/m7y2016_knn.pkl

95.8 s, 16:05:20 ,t0, 16:03:44

tmp/m7y2016_forest.pkl
103.9 s, 16:07:04 ,t0, 16:05:20

tmp/m7y2016_gbdtd.pkl
```

```
2041.58 s, 16:41:05 ,t0, 16:07:04
```

```
tmp/m7y2016_dtree.pkl
```

```
22.12 s, 16:41:27 ,t0, 16:41:05
```

```
tmp/m7y2016_mlp.pkl
```

```
306.79 s, 16:46:34 ,t0, 16:41:27
```

```
tmp/m7y2016_mlpreg.pkl
```

```
54.15 s, 16:47:28 ,t0, 16:46:34
```

如图 14-6 所示是构建累计多字段足彩赔率模型时各种机器学习算法的运算时间对比。

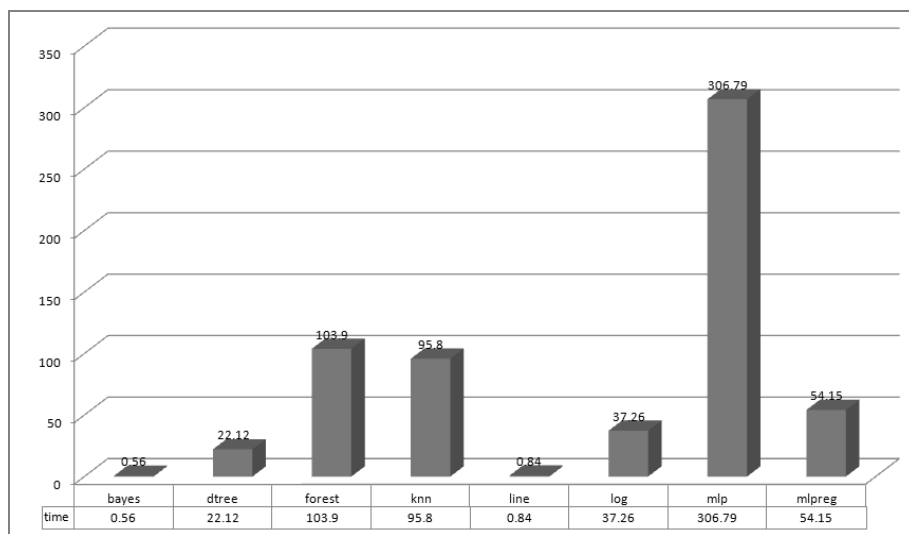


图 14-6 机器学习算法构建累计多字段足彩赔率模型的时间对比

由图 14-6 和案例输出信息可以看出：

- 线性回归算法模型构建速度最快，才 0.69 秒；
- GBDT 迭代决策树算法耗时最长，使用 2041 秒约 35 分钟，比其他各种算法的总时间还长；
- MLP 神经网络算法和 MLPreg 神经网络算法的模型构建时间分别是 306 秒和 54 秒。

生成的足彩模型文件保存在 tmp 目录下，需要人工复制到目录/tfbDat/mlib/中。

14.3.3 足彩算法模型文件

以上案例生成的足彩模型文件都保存在 tmp 目录下，需要人工复制到目录/tfbDat/mlib/中。

在 tfb_sys 模块中，设置了相关路径的全局变量，代码如下：

```
rdat0='/tfbDat/'
rxdat=rdat0+'xdat/'
rmdat=rdat0+'mdat/'
rmlib=rdat0+'mlib/' #ai.mx.lib.xxx
```

其中：

- rdat0 为足彩数据根目录；
- rxdat 为 pnv 数据文件目录；
- rmdat 为赔率数据合并后的文件目录；
- rmlib 为足彩模型算法文件目录。

如图 14-7 所示是 mlib 目录下的算法模型文件。

名称	大小		
m3y2016_bayes.pkl	1 KB	p3y2016_line.pkl	1 KB
m3y2016_dtree.pkl	57,057 KB	p3y2016_log.pkl	1 KB
m3y2016_forest.pkl	355,738 KB	p3y2016_mlp.pkl	27 KB
m3y2016_gbdtpkl	1,023 KB	p3y2016_mlpreg.pkl	21 KB
m3y2016_knn.pkl	142,230 KB	p7y2016_bayes.pkl	2 KB
m3y2016_line.pkl	1 KB	p7y2016_dtree.pkl	23,262 KB
m3y2016_log.pkl	1 KB	p7y2016_forest.pkl	156,435 KB
m3y2016_mlp.pkl	27 KB	p7y2016_gbdtpkl	1,023 KB
m3y2016_mlpreg.pkl	21 KB	p7y2016_knn.pkl	48,461 KB
m7y2016_bayes.pkl	2 KB	p7y2016_line.pkl	1 KB
m7y2016_dtree.pkl	133,644 KB	p7y2016_log.pkl	2 KB
m7y2016_forest.pkl	885,407 KB	p7y2016_mlp.pkl	40 KB
m7y2016_gbdtpkl	1,021 KB	p7y2016_mlpreg.pkl	33 KB
m7y2016_knn.pkl	291,209 KB	pzy2016_bayes.pkl	2 KB
m7y2016_line.pkl	1 KB	pzy2016_dtree.pkl	4,367 KB
m7y2016_log.pkl	2 KB	pzy2016_forest.pkl	92,946 KB
m7y2016_mlp.pkl	40 KB	pzy2016_gbdtpkl	1,029 KB
m7y2016_mlpreg.pkl	33 KB	pzy2016_knn.pkl	154,058 KB
p3y2016_bayes.pkl	1 KB	pzy2016_line.pkl	1 KB
p3y2016_dtree.pkl	14,351 KB	pzy2016_log.pkl	2 KB
p3y2016_forest.pkl	91,520 KB	pzy2016_mlp.pkl	93 KB
p3y2016_gbdtpkl	1,018 KB	pzy2016_mlpreg.pkl	86 KB
p3y2016_knn.pkl	23,615 KB		

图 14-7 mlib 目录下的算法模型文件

由图 14-7 mlib 目录下的算法模型文件可以看出：

- 文件最大的是 m7year2016_forest.pkl，大于 800MB，虽然是累计赔率模型，只有 7 个数据字段，却比合并后的总赔率数据文件 xdat2017（约 400MB）大一倍左右；
- 全字段累积赔率模型，按照 7 字段累积赔率文件与 3 字段累加模型文件的对比，文件大小应该在 2GB 左右；
- 文件大小最小的 Line 线性回归算法模型文件、Log 逻辑回归算法模型文件和 Bayes 贝叶斯算法模型文件都只有 1KB 左右；
- GBDT 迭代回归算法模型虽然运算时间最长，但模型文件大小却只有 1MB 左右，有趣的是 GBDT 3 字段的模型文件的大小和 7 字段的模型文件的大小差不多，甚至还小 2KB，可见算法模型文件的大小和数据字段并非完全是正比关系；
- 其他文件大小较大的模型还有 KNN（K 近邻）算法模型和 dtree 决策树算法模型。

15

第 15 章

足彩机器学习模型验证

前面介绍了各种机器学习算法模型的构建并保存在以下目录：

/tfbDat/mlib/

本章使用 2017 年最新的足彩实盘数据，验证相关模型的准确度。

15.1 年度赔率模型验证

年度赔率模型验证是指测试的机器学习算法模型都是基于一年的单个年度的算法模型，简而言之，就是算法模型文件名是以字母 p 开头的文件。

15.1.1 案例 15-1：年度赔率模型验证

案例 15-1 的文件名是 zd201_mchk01.py，讲解验证年度赔率模型，下面分组介绍相关的程序代码。

第 1 组代码，设置相关参数和算法名称列表：

```
#1
rs0,ysgn='/tfbDat/','kwin'
fsr=rs0+'mdat/year_2017.dat'
mxlst=['line','log','bayes','knn','forest','gbdt','dtree','mlp','mlpr
eg']
```

```
print('#1',fsr)
```

在以上代码当中：

- ysgn 是结果数据的字段名称，不同项目采用的名称是不同的；
- 验证数据使用的是 year_2017.dat 年度数据，虽然 2017 年只有 2 个月，不过也有近千场比赛数据，足够进行验证测试了；
- mxlst 机器学习算法名称类别删除了最耗时间的 SVM 向量机机器学习算法模型。

第 2 组代码，设置机器学习算法文件参数：

```
#2
print('#2 xlst')
fsr0=rs0+'mllib/p3y2016_'
xlst=['pwin0','pdraw0','plost0']
```

其中：

- fsr0 是机器学习算法文件名前缀。
- xlst 是与机器学习算法对应的数据字段列表名称。

第 3 组代码，读取测试数据集：

```
#3
print('#2 x_dat,y_dat')
x_dat,y_dat= tft.fb_xdat_xrd020(fsr,xlst,ysgn,1,True)
```

第 4 组代码，批量读取机器学习算法模型文件，并保存到全局变量 zai.xmodel：

```
#4
zai.xmodel={}
print('\n#4,ai_f_mxRdlst')
zai.ai_f_mxRdlst(fsr0,mxlst)
```

调用前，注意清空机器学习算法模型库的全局变量 zai.xmodel。

第 5 组代码，使用 zai.mx_funlst8mx 批量算法模型测试函数，测试各个模型库的准确度：

```
#5
print('\n#5,mx_funlst8mx')
zai.mx_funlst8mx(mxlst, x_dat, y_dat,yk0=1,fgInt=True)
```

案例 15-1 的主要目的就是得到第 5 组代码的输出结果，对应的输出信息是：

```
#5,mx_funlst8mx
ok:28.87%,mx,line,tn,0.07 s
ok:52.97%,mx,log,tn,0.01 s
ok:53.69%,mx,bayes,tn,0.01 s
```

```

ok:44.48%,mx,knn,tn,0.37 s
ok:46.96%,mx,forest,tn,0.10 s
ok:53.71%,mx,gdbt,tn,0.29 s
ok:45.53%,mx,dtree,tn,0.02 s
ok:53.69%,mx,mlp,tn,0.05 s
ok:37.52%,mx,mlpreg,tn,0.05 s

```

如图 15-1 所示是各种机器学习算法的准确度比较。

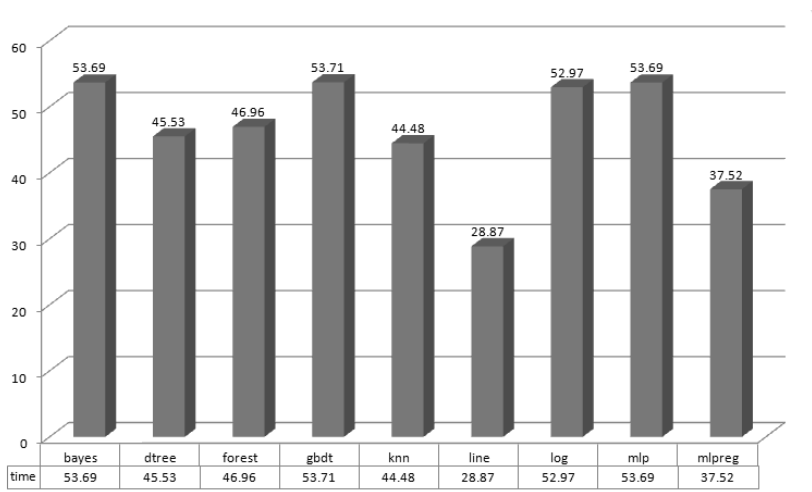


图 15-1 各种机器学习算法的准确度比较

由图 15-1 和案例输出信息可以看出：

- 尽管各个机器学习算法模型构建的运算时间相差很远，但通过模型分析数据预测结果，发现时间都在 0.5 秒以内，非常迅速。
- 以上机器学习算法模型使用的数据源是 2017 年 1—2 月的足彩赔率数据。
- 准确度最高的是 Log 回归算法模型、Bayes 贝叶斯算法模型、GBDT 迭代决策树模型和 MLP 神经网络模型，准确度在 53% 左右。
- 准确度最低的是 Line 线性回归模型，只有 28.87%。

15.1.2 案例 15-2：多字段年度赔率模型验证

案例 15-2 的文件名是 zd202_mchk02.py，介绍验证多字段年度赔率模型，除了

第 2 组代码，其他代码与案例 15-1 完全一样。

案例 15-1 的第 2 组代码用于设置模型的相关参数：

```
#2
print('#2 xlst')
fsr0=rs0+'mlib/p3y2016_'
xlst=['pwin0','pdraw0','plost0']
```

其中，xlst 变量只使用 3 个最核心的字段，即开盘胜、平、负的赔率，所以模型文件名以 p3 开头。

案例 15-2 的第 2 组代码是：

```
#2
print('#2 xlst')
fsr0=rs0+'mlib/p7y2016_'
xlst=['cid','pwin0','pdraw0','plost0','pwin9','pdraw9','plost9']
```

案例 15-2 的第 2 组代码使用的是 7 字段参数设置，采用的赔率模型文件名开头是 p7y2016_。

xlst 使用的数据字段名包括：博彩机构代码，开盘胜、平、负赔率，收盘胜、平、负赔率等数据。

案例 15-2 部分运行结果如下：

```
#5,mx_funlst8mx
ok:29.19%,mx,line,tn,0.02 s
ok:53.47%,mx,log,tn,0.02 s
ok:49.85%,mx,bayes,tn,0.01 s
ok:45.42%,mx,knn,tn,1.29 s
ok:45.14%,mx,forest,tn,0.11 s
ok:53.63%,mx,gbdt,tn,0.29 s
ok:42.33%,mx,dtree,tn,0.03 s
ok:52.59%,mx,mlp,tn,0.10 s
ok:34.81%,mx,mlpreg,tn,0.05 s
```

如图 15-2 所示是各种机器学习算法的准确度比较。

由图 15-2 和案例输出信息可以看出：

- 尽管各个机器学习算法模型构建的运算时间相差很远，但通过模型分析数据预测结果，发现时间都很短，除了 KNN（K 近邻）机器学习算法，其他算法运行时间都在 0.5 秒以内，非常迅速；

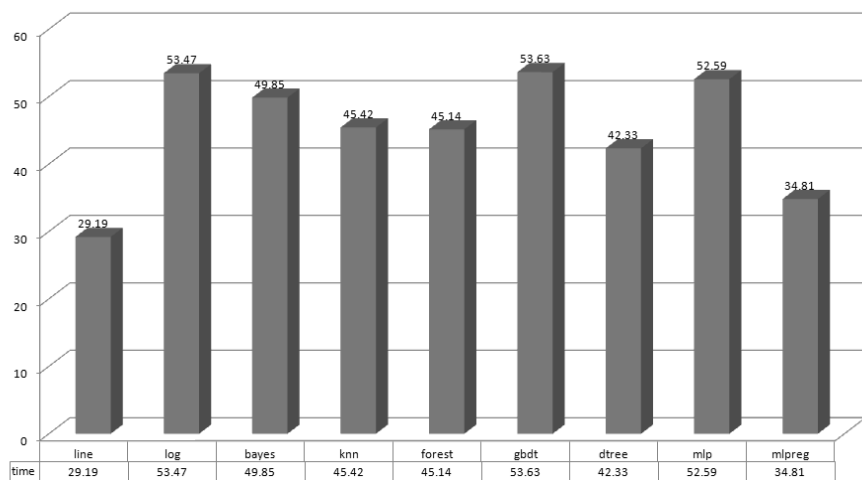


图 15-2 各种机器学习算法的准确度比较

- 以上机器学习算法模型使用的数据源是 2017 年 1—2 月的足彩赔率数据；
- 准确度最高的是 Log 回归算法模型和 GBDT 迭代决策树模型，准确度在 53% 左右；
- 准确度最低的是 Line 线性回归模型，只有 29.19%。

以 pzy2016_开头的文件名是全字段机器学习算法模型，xlst 使用的数据字段名包括：博彩机构代码，开盘胜、平、负赔率，收盘胜、平、负赔率等数据。

全字段模型需要进一步对数据进行清洗，所以我们暂时省略了相关的案例。

15.2 累计赔率模型验证

累计赔率模型验证是指测试的机器学习算法模型都是基于多年累计的算法模型，简而言之，就是算法模型文件名是以字母 m 开头的文件。

15.2.1 案例 15-3：累计赔率模型验证

案例 15-3 的文件名是 zd203_mchk11.py，介绍验证累计赔率模型，案例 15-3 除了第 2 组代码，其他代码与案例 15-1 完全一样。

案例 15-1 的第 2 组代码用于设置模型相关参数：

```
#2
print('#2 xlst')
fsr0=rs0+'mlib/p3y2016_'
xlst=['pwin0','pdraw0','plost0']
```

其中，`xlst` 变量只使用 3 个最核心的字段，即开盘胜、平、负的赔率，所以模型文件名以 `p3` 开头。

案例 15-3 的第 2 组代码是：

```
#2
print('#2 xlst')
fsr0=rs0+'mlib/m3y2016_'
xlst=['pwin0','pdraw0','plost0']
```

案例 15-3 的第 2 组代码使用的虽然也是 3 字段参数设置，但采用的赔率模型文件名开头是 `m3y2016_`，表示是累计赔率模型文件。

15.2.2 案例 15-4：多字段累计赔率模型验证

案例 15-4 的文件名是 `zd204_mchk12.py`，介绍验证多字段累计赔率模型。

案例 15-1 的第 2 组代码用于设置模型相关参数：

```
#2
print('#2 xlst')
fsr0=rs0+'mlib/p3y2016_'
xlst=['pwin0','pdraw0','plost0']
```

其中，`xlst` 变量只使用了 3 个最核心的字段，即开盘胜、平、负的赔率，所以模型文件名以 `p3` 开头。

案例 15-4 的第 2 组代码是：

```
#2
print('#2 xlst')
fsr0=rs0+'mlib/m7y2016_'
xlst=['cid','pwin0','pdraw0','plost0','pwin9','pdraw9','plost9']
```

案例 15-4 的第 2 组代码使用的是 7 字段参数设置，同时使用的赔率模型文件名开头是 `m7y2016_`，表示是累计赔率模型文件。

案例 15-3 与案例 15-4 类似，所以对其输出数据进行了合并处理，如图 15-3 所示。

<pre>#5,mx_funlst8mx ok:28.96%,mx,line,tn,0.01 s ok:53.09%,mx,log,tn,0.01 s ok:53.63%,mx,bayes,tn,0.01 s ok:44.88%,mx,knn,tn,0.76 s ok:49.01%,mx,forest,tn,0.13 s ok:53.68%,mx,gbdt,tn,0.29 s ok:48.15%,mx,dtree,tn,0.03 s ok:53.63%,mx,mlp,tn,0.07 s ok:34.28%,mx,mlpreg,tn,0.05 s</pre> <p>案例 15-3 3 字段输出数据</p>	<pre>#5,mx_funlst8mx ok:29.56%,mx,line,tn,0.02 s ok:14.13%,mx,log,tn,0.01 s ok:10.71%,mx,bayes,tn,0.01 s ok:20.62%,mx,knn,tn,2.52 s ok:18.87%,mx,forest,tn,0.21 s ok:15.00%,mx,gbdt,tn,0.29 s ok:17.91%,mx,dtree,tn,0.03 s ok:12.06%,mx,mlp,tn,0.11 s ok:39.90%,mx,mlpreg,tn,0.06 s</pre> <p>案例 15-4 7 字段输出数据</p>
--	--

图 15-3 3 字段和 7 字段的输出数据

如图 15-4 所示是各种机器学习算法的准确度比较。

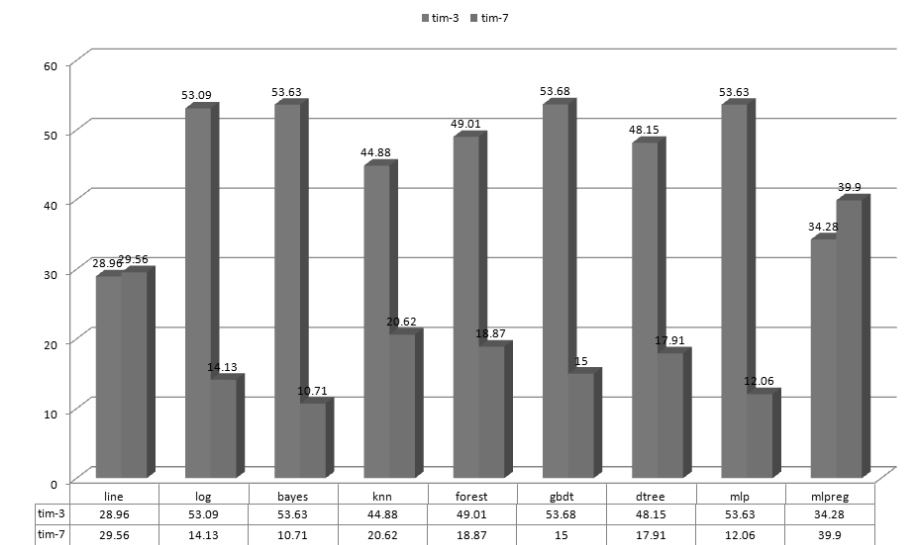


图 15-4 各种机器学习算法的准确度比较

由图 15-4 和案例输出信息可以看出：

- 在各个机器学习算法模型中，7 字段模型（tim-7）的数据分析时间大约比 3 字段模型（tim-3）的数据分析时间（tim-3）慢一倍；
- 整体的运算时间都很快，除了 KNN（K 近邻）机器学习算法，其他算法的运行

时间都在 0.5 秒以内，非常迅速；

- 以上机器学习算法模型使用的数据源是 2017 年 1—2 月的足彩赔率数据；
- 准确度最高的是 Log 回归算法模型和 GBDT 迭代决策树模型，准确度在 53% 左右；
- 准确度最低的是 Line 线性回归模型，只有 29% 左右。

需要注意的是，除了 Line 线性回归模型和 MLP 神经网络模型，其他 3 字段模型的准确度均高于 7 字段模型的准确度。

以上数据，再一次证明了笔者的小数据理论：

人工智能，机器学习，不是数据量越大越好，也不是迭代计算的次数越多越好。

15.3 年度组合模型验证

年度组合模型验证是使用多种单个年度的机器学习算法模型组成的算法合集，验证最终的结果。

15.3.1 案例 15-5：年度组合模型验证

案例 15-5 的文件名是 zd205_mul01.py，介绍年度组合模型验证程序，下面逐一代码进行介绍。

第 1 组代码，设置验证参数：

```
#1
rs0,ysgn='/tfbDat/','kwin'
fsr=rs0+'mdat/year_2017.dat'
mxlst=['line','log','bayes','knn','forest','gbdt','dtree','mlp','mlpr
eg']

mlst0=mxlst
mlst1=['log','gbdt','mlp']
mlst2=['log','gbdt']
mlst3=['log','mlp']
mlst4=['gbdt','mlp']
mlst5=['forest','dtree']
```

```
print('#1',fsr)
```

其中：

- fsr 是测试数据文件 year_2017.dat 在 2017 年 1—2 月的足彩赔率数据；
- mxlst 是机器学习算法的名称列表；
- mlst0~mlst5 是不同的机器学习算法组合，mlst0=mxlst 表示是所有的模型。

第 2 组代码用于进行模型参数验证：

```
#2
print('#2 xlst')
fsr0=rs0+'mllib/p3y2016_'
xlst=['pwin0','pdraw0','plost0']
```

第 2 组代码使用的是 3 字段的年度机器学习算法模型文件。

第 3 组代码，读取测试数据集：

```
#3
print('#2 x_dat,y_dat')
x_dat,y_dat= tft.fb_xdat_xrd020(fsr,xlst,ysgn,1,True)
```

第 4 组代码，读取各种机器学习算法模型，模型库保存到全局变量 zai.xmodel：

```
#4
zai.xmodel={}
print('\n#4,ai_f_mxRdlst')
zai.ai_f_mxRdlst(fsr0,mxlst)
```

第 5 组至第 9 组代码基本一样，就是调用各种不同的 mlst 机器学习算法组合，测试组合算法的精确度：

```
#5
print('\n#5,mlst0')
zai.mx_mul(mlst0, x_dat, y_dat,yk0=1,fgInt=True)
```

第 5 组代码的输出信息如下：

```
y_pred,预测
1 y_pred01,kok:28.87% line 0.0 s
2 y_pred02,kok:52.97% log 0.0 s
3 y_pred03,kok:53.69% bayes 0.0 s
4 y_pred04,kok:44.48% knn 0.36 s
5 y_pred05,kok:46.96% forest 0.09 s
6 y_pred06,kok:53.71% gbdn 0.28 s
7 y_pred07,kok:45.53% dtree 0.01 s
8 y_pred08,kok:53.69% mlp 0.04 s
```

```
9 y_pred09,kok:37.52% mlpreg 0.04 s
@mx:mx_sum,kok:49.42%
```

如图 15-5 所示是各种机器学习算法的准确度对比。

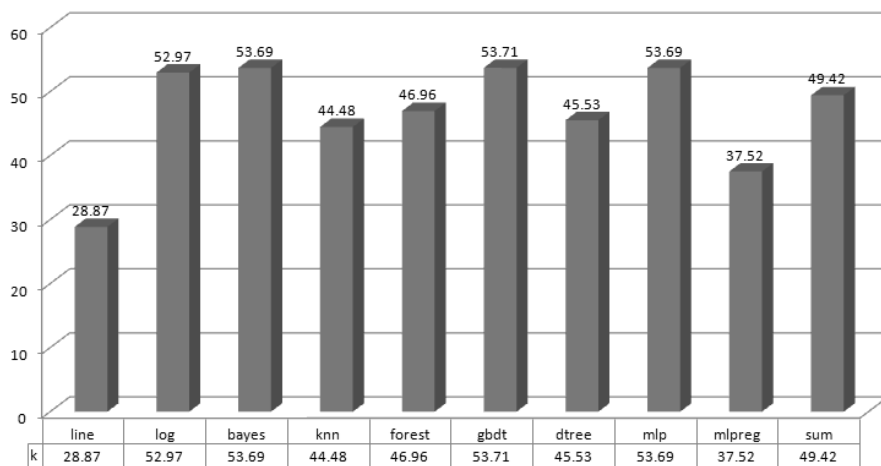


图 15-5 各种机器学习算法的准确度对比

由图 15-5 和输出信息可以看出，最终的机器学习组合算法（sum）的准确度只有 49.4%，低于组合当中多种单一机器学习算法的准确度。

在案例 15-5 的各组机器算法组合中，只有 mlst1 和 mlst2 算法组合的准确度略微高于其中最高单一算法：

```
#6,mlst1
y_pred,预测
1 y_pred01,kok:52.97% log 0.0 s
2 y_pred02,kok:53.69% bayes 0.0 s
3 y_pred03,kok:53.71% gbd 0.28 s
4 y_pred04,kok:53.69% mlp 0.04 s
@mx:mx_sum,kok:53.82%

#7,mlst2
y_pred,预测
1 y_pred01,kok:44.48% knn 0.35 s
2 y_pred02,kok:46.96% forest 0.09 s
3 y_pred03,kok:45.53% dtree 0.01 s
```

```
@mx:mx_sum,kok:44.59%
```

其他各种机器算法的组合输出信息，请读者自己参看程序输出信息。

15.3.2 案例 15-6：多字段年度组合模型验证

案例 15-6 的文件名是 `zd206_mul02.py`，介绍多字段年度组合模型验证程序。

案例 15-6 与案例 15-5 只是第 2 组代码略有不同，案例 15-6 中设置模型参数验证的代码如下：

```
#2
print('#2 xlst')
fsr0=rs0+'mlib/p7y2016_'
xlst=['cid','pwin0','pdraw0','plost0','pwin9','pdraw9','plost9']
```

案例 15-5 使用的是 3 字段的年度机器学习算法模型文件。

案例 15-6 使用的是 7 字段的年度机器学习算法模型文件。

15.3.3 案例 15-7：全字段年度组合模型验证

案例 15-7 的文件名是 `zd207_mul03.py`，介绍全字段年度组合模型验证程序。

案例 15-7 与案例 15-5 只是第 2 组代码略有不同，案例 15-7 中设置模型参数验证的代码如下：

```
#2
print('#2 xlst')
fsr0=rs0+'mlib/pzy2016_'
xlst=['cid','pwin0','pdraw0','plost0','pwin9','pdraw9','plost9'
      ,'vwin0','vdraw0','vlost0','vwin9','vdraw9','vlost9','vback0','v'
back9'
      ,'vwin0kali','vdraw0kali','vlost0kali','vwin9kali','vdraw9kali',
'vlost9kali'
      ,'mtid','gtid','tweek']
```

案例 15-5 使用的是 3 字段的年度机器学习算法模型文件。

案例 15-7 使用的是全字段的年度机器学习算法模型文件。

15.3.4 年度组合模型测试数据对比分析

案例 15-6 与案例 15-7 的输出信息差不多，如图 15-6 所示是两个案例的输出信息对比分析。

<pre> 1 y_pred01,kok:29.19% line 0.0 s 2 y_pred02,kok:53.47% log 0.0 s 3 y_pred03,kok:49.85% bayes 0.0 s 4 y_pred04,kok:45.42% knn 1.28 s 5 y_pred05,kok:45.14% forest 0.1 s 6 y_pred06,kok:53.63% gbdn 0.29 s 7 y_pred07,kok:42.33% dtree 0.01 s 8 y_pred08,kok:52.59% mlp 0.09 s 9 y_pred09,kok:34.81% mlpreg 0.04 s @mx:mx_sum,kok:47.59% </pre> <p>案例 15-6 7 字段测试数据</p>	<pre> 1 y_pred01,kok:37.63% line 0.01 s 2 y_pred02,kok:53.31% log 0.01 s 3 y_pred03,kok:34.18% bayes 0.01 s 4 y_pred04,kok:43.08% knn 1.23 s 5 y_pred05,kok:45.74% forest 0.09 s 6 y_pred06,kok:52.52% gbdn 0.3 s 7 y_pred07,kok:42.85% dtree 0.01 s 8 y_pred08,kok:53.00% mlp 0.26 s 9 y_pred09,kok:33.36% mlpreg 0.04 s @mx:mx_sum,kok:47.22% </pre> <p>案例 15-7 全字段测试数据</p>
--	--

图 15-6 机器学习算法结果对比

如图 15-7 所示的机器学习算法结果对比图中还增加了案例 15-5 的 3 字段测试数据。

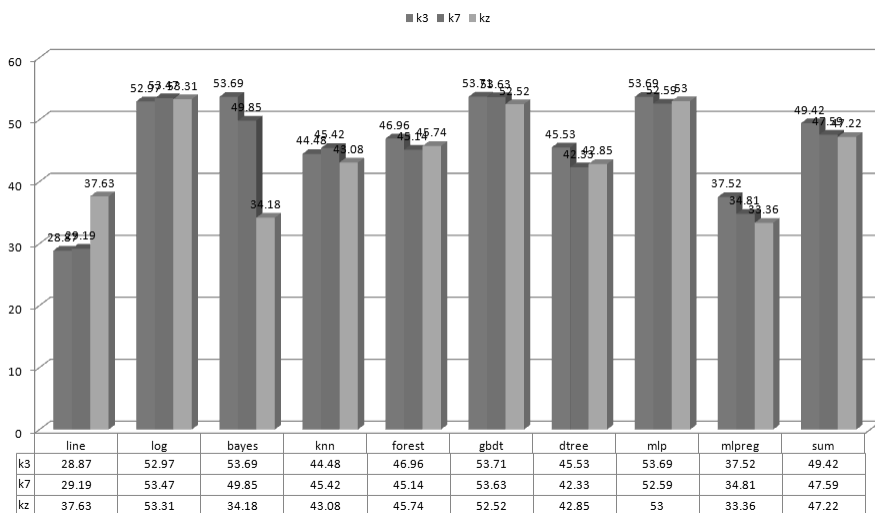


图 15-7 机器学习算法结果对比图

由图 15-6 和图 15-7 可以看出：

- 同一个机器学习算法模型，其 3 字段（k3）、7 字段（k7）和全字段（kz）的算法准确度都差不多，只有 Line 线性回归算法和 Bayes 贝叶斯算法部分字段模型差别较大。
- 准确度最高的是 Log 逻辑回归算法、GBDT 迭代决策树算法和 MLP 神经网络算法。
- 各个字段的组合算法总的准确度，均低于组合当中的最高单一算法的准确度。

15.4 累计组合模型验证

累计组合模型验证是使用多种累计年度的机器学习算法模型组成的算法合集，用来验证最终的结果。

15.4.1 案例 15-8：年度组合模型验证

案例 15-8 介绍年度组合模型验证程序，案例 15-8 与案例 15-5 只是第 2 组代码参数设置有所不同。

案例 15-5 的程序文件名是 zd405zd205_mul01.py，第 2 组代码是：

```
#2
print('#2 xlst')
fsr0=rs0+'mlib/p3y2016_'
xlst=['pwin0','pdraw0','plost0']
```

案例 15-8 的文件名是 zd208_mm01.py，第 2 组代码是：

```
#2
print('#2 xlst')
fsr0=rs0+'mlib/m3y2016_'
xlst=['pwin0','pdraw0','plost0']
```

对比以上代码，会发现案例 15-5 使用的数据源文件前缀以字母 p 开头，表示基于单一年度赔率数据的机器学习算法模型：

```
fsr0=rs0+'mlib/p3y2016_'
```

而案例 15-8 使用的数据源文件前缀以字母 m 开头,表示基于多年累计赔率数据的机器学习算法模型:

```
fsr0=rs0+'mlib/m3y2016_'
```

案例 15-8 与案例 15-5 的其他代码完全一致。

15.4.2 案例 15-9: 多字段年度组合模型验证

案例 15-9 介绍多字段年度组合模型验证程序。案例 15-9 与案例 15-6 只是第 2 组代码的参数设置有所不同。

案例 15-6 的程序文件名是 zd405zd206_mul02.py, 第 2 组代码是:

```
#2
print('#2 xlst')
fsr0=rs0+'mlib/p7y2016_'
xlst=['cid','pwin0','pdraw0','plost0','pwin9','pdraw9','plost9']
```

案例 15-9 的文件名是 zd209_mm02.py, 第 2 组代码是:

```
#2
print('#2 xlst')
fsr0=rs0+'mlib/m7y2016_'
xlst=['cid','pwin0','pdraw0','plost0','pwin9','pdraw9','plost9']
```

对比以上代码,会发现案例 15-6 使用的数据源文件前缀以字母 p 开头,表示基于单一年度赔率数据的机器学习算法模型:

```
fsr0=rs0+'mlib/p7y2016_'
```

而案例 15-9 使用的数据源文件前缀以字母 m 开头,表示基于多年累计赔率数据的机器学习算法模型:

```
fsr0=rs0+'mlib/m7y2016_'
```

案例 15-9 与案例 15-6 的其他代码完全一致。

15.4.3 累计组合模型测试数据对比分析

案例 15-8 与案例 15-9 的输出信息差不多,如图 15-8 所示是两个案例的输出信息对比分析。


```

1 y_pred01,kok:28.96% line 0.0 s
2 y_pred02,kok:53.09% log 0.0 s
3 y_pred03,kok:53.63% bayes 0.0 s
4 y_pred04,kok:44.88% knn 0.76 s
5 y_pred05,kok:49.01% forest 0.12 s
6 y_pred06,kok:53.68% gbdt 0.28 s
7 y_pred07,kok:48.15% dtree 0.02 s
8 y_pred08,kok:53.63% mlp 0.06 s
9 y_pred09,kok:34.28% mlpreg 0.04 s
@mx:mx_sum,kok:50.79%

```

案例 15-8 3 字段测试数据

```

1 y_pred01,kok:29.56% line 0.0 s
2 y_pred02,kok:14.13% log 0.0 s
3 y_pred03,kok:10.71% bayes 0.0 s
4 y_pred04,kok:20.62% knn 2.63 s
5 y_pred05,kok:18.87% forest 0.2 s
6 y_pred06,kok:15.00% gbdt 0.28 s
7 y_pred07,kok:17.91% dtree 0.03 s
8 y_pred08,kok:12.06% mlp 0.1 s
9 y_pred09,kok:39.90% mlpreg 0.04 s
@mx:mx_sum,kok:36.81%

```

案例 15-9 多字段测试数据

图 15-8 机器学习算法测试数据对比

如图 15-9 所示是机器学习算法测试数据对比图，汇总了案例 15-5 到案例 15-9 的测试数据。

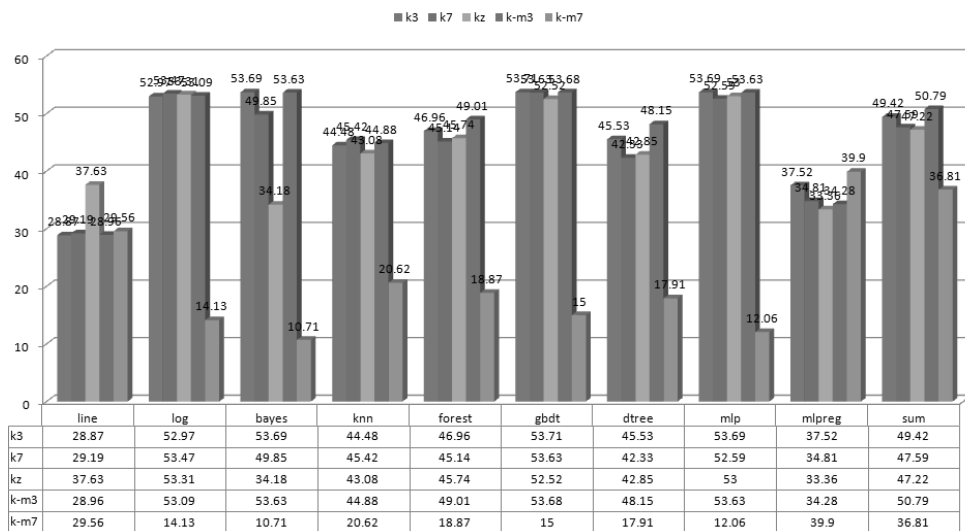


图 15-9 机器学习算法结果对比图

由图 15-8 和图 15-9 可以看出：

- k3、k7、kz 分别是机器学习算法单一年度赔率数据 3 字段、7 字段和全字段模型的测试数据；

- k-m3 和 k-m7 分别是机器学习算法多年累计赔率数据的 3 字段、7 字段模型测试数据；
- 7 字段模型测试数据与其他模型测试数据差别较大；
- 删除 k-m7 的 7 字段模型测试数据，在其他测试模型中，Log 逻辑回归算法、GBDT 迭代决策树算法和 MLP 神经网络算法测试数据的准确度都在 50%以上；
- Line 线性回归算法，虽然整体测试准确度偏低，但 k-m7 的 7 字段模型测试数据的准确度居于第二；
- 各个字段的组合算法总的准确度，均低于组合当中最高单一算法的准确度。

16

第 16 章

结果数据分析

16.1 神秘的df9

在之前的案例中，通常把最终结果数据保存在 df9 变量中，本章将具体讲解变量 df9 的来龙去脉。

16.1.1 案例 16-1：调试模式

案例 16-1 的文件名是 zd301_df01.py，介绍使用调试模式 fgDebug，以及查看最终结果数据 df9，下面分组讲解程序源码。

第 1 组代码，设置参数，测试数据文件是 year_2017.dat，即 2017 年的赔率数据。

```
#1
rs0,ysgn='/tfbDat/', 'kwin'
fsr=rs0+'mdat/year_2017.dat'
mxlst=['line']
print('#1',fsr)
```

第 2 组代码，设置机器学习算法模型文件参数，使用的是 3 字段的年度模型文件：

```
#2
print('#2 xlst')
fsr0=rs0+'mllib/p3y2016_'
xlst=['pwin0','pdraw0','plost0']
```

第 3 组代码，读取测试数据集：

```
#3
print('#2 x_dat,y_dat')
x_dat,y_dat=tft.fb_xdat_xrd020(fsr,xlst,ysgn,1,True)
```

第 4 组代码，读取机器学习算法模型文件组，并保存到全局变量 `zai.xmodel` 作为系统的模型库：

```
#4
zai.xmodel={}
print('\n#4,ai_f_mxRdlst')
zai.ai_f_mxRdlst(fsr0,mxlst)
```

第 5 组代码，设置机器学习算法代码 `msgn` 和变量 `mx`，调用 `zai.mx_fun8mx` 机器学习接口函数：

```
#5
print('\n#5,mx_funlst8mx')
msgn=mxlst[0]
mx=zai.xmodel[msgn]
dacc,df9=zai.mx_fun8mx(mx, x_dat,
y_dat,yk0=1,fgInt=True,fgDebug=True)
```

以上代码都很简单，需要注意的只有第 5 组代码的最后一行：

```
dacc,df9=zai.mx_fun8mx(mx, x_dat,
y_dat,yk0=1,fgInt=True,fgDebug=True)
```

在调用 `zai.mx_fun8mx` 机器学习接口函数时，设置参数 `fgDebug=True`，这里首次使用了调试模式。此外，在函数调用左侧，还有两个返回变量：`dacc` 准确度和神秘的 `df9`。

以下是在 `fgDebug=True` 调试模式下，第 5 组代码对应的输出信息：

```
#5,mx_funlst8mx

ai_acc_xed
    pwin0  pdraw0  plostd0  y_test  y_pred  y_preds  ysub  ysub2
y_test_div  ysubk
    0  1.24  4.90  8.5  2  1  1.409777  1  1  2.0  50.0
```

1	1.29	5.50	10.0	2	1	1.467774	1	1	2.0	50.0
2	1.25	5.15	9.0	2	1	1.430041	1	1	2.0	50.0
3	1.30	5.25	10.0	2	1	1.464406	1	1	2.0	50.0
4	1.29	5.50	9.5	2	1	1.449774	1	1	2.0	50.0

```
n_df9,34706,n_dfk,10021
```

```
acc-kok: 28.87%, MAE:0.71, MSE:0.72, RMSE:0.85
```

以上输出信息,均源自 `ztop_ai` 极宽智能模块库中的 `zai.ai_acc_xed` 结果验证函数的第 3 组代码:

```
#3
if fgDebug:
    print('\nai_acc_xed')
    print(df9.head())
    y_test,y_pred=df9['y_test'],df9['y_pred']
    print('\ntest, {0}, npred, {1}, dsum, {2}'.format(ny_test,ny_pred,
dsum))

    dmae=metrics.mean_absolute_error(y_test, y_pred)
    dmse=metrics.mean_squared_error(y_test, y_pred)
    drmse=np.sqrt(metrics.mean_squared_error(y_test, y_pred))
    print('acc-kok: {0:.2f}%, MAE:{1:.2f}, MSE:{2:.2f},
RMSE:{3:.2f}'.format(dacc,dmae,dmse,drmse))
```

首先输出结果变量 `df9` 的头部数据,显示结果数据中的字段结构和部分信息。

最后两行是一些分析数据:

```
n_df9,34706,n_dfk,10021
```

```
acc-kok: 28.87%, MAE:0.71, MSE:0.72, RMSE:0.85
```

其中:

- `nd_df9` 为结果数据总长度, `n_dfk` 为预测正确的数目;
- `acc-kok` 为正确率百分比;
- `MAE` 为平均绝对误差;
- `MSE` 为均方差、方差,结果数据越接近于零,说明模型选择和拟合越好,数据预测也越成功;
- `RMSE` 为均方根、标准差,结果数据越接近于零,说明模型选择和拟合越好,数据预测也越成功。

16.1.2 神秘的 df9 结果数据变量

下面来看一下神秘的 df9。结果数据变量 df9 使用的是 Pandas 的 DataFrame 数据格式，在调试模式下，会在 tmp 目录下生成一个 df9_pred.csv 文件，对应的就是 df9 的内容。

df9 变量源自 zai.mx_fun8mx 等机器学习算法的调用，参见以下代码：

```
def mx_fun8mx(mx,x_test,y_test,yk0=5,fgInt=False,fgDebug=False):
    #1
    df9=x_test.copy()
```

本案例中，df9 变量最早是 x_test 数据的复制品，然后在 mx_fun8mx 函数运算中增加了部分中间字段。

如图 16-1 所示是用 Excel 打开的结果数据文件截图。

	A	B	C	D	E	F	G	H	I	J	K
1	pwin0	pdraw0	plost0	y_test	y_pred	y_preds	ysub	ysub2	y_test_dysubk		
2	1.24	4.9	8.5	2	1	1.409777	1	1	2	50	
3	1.29	5.5	10	2	1	1.467774	1	1	2	50	
4	1.25	5.15	9	2	1	1.430041	1	1	2	50	
5	1.3	5.25	10	2	1	1.464406	1	1	2	50	
6	1.29	5.5	9.5	2	1	1.449774	1	1	2	50	
7	1.3	5.5	9.5	2	1	1.449222	1	1	2	50	
8	1.3	5.25	10	2	1	1.464406	1	1	2	50	
9	1.28	5.9	10	2	1	1.472831	1	1	2	50	
10	1.27	5.8	8.3	2	1	1.411058	1	1	2	50	
11	1.33	5.25	10.5	2	1	1.48075	1	1	2	50	
12	1.25	4	7	2	1	1.34509	1	1	2	50	
13	1.3	5.25	9	2	1	1.428407	1	1	2	50	
14	1.3	5.25	9	2	1	1.428407	1	1	2	50	
15	1.33	5.75	9.65	2	1	1.455782	1	1	2	50	
16	1.29	5	10	2	1	1.462143	1	1	2	50	
17	1.27	5.8	7	2	1	1.364259	1	1	2	50	
18	1.3	5.5	9.5	2	1	1.449222	1	1	2	50	
19	1.28	6	11	2	2	1.509956	0	0	2	0	
20	1.27	5.5	7.6	2	1	1.38248	1	1	2	50	

图 16-1 结果数据文件截图

需要说明的是，图 16-1 只是本案例和 zai.mx_fun8mx 机器学习接口函数中，单一机器学习算法的 df9 结果数据截图，如果是 zai.mx_mul 机器学习组合算法等函数，还会增加几个以 y_pred 开头带数字后缀的数据列，表示不同算法的预测结果。

图 16-1 中的 df9 结果数据，部分由 zai.ai_acc_xed 结果验证函数生成，以下是部分相关的函数代码：

```
#1
ny_test,ny_pred=len(df9['y_test']),len(df9['y_pred'])
ny_test=len(df9['y_test'])
df9['ysub']=df9['y_test']-df9['y_pred']
```

```

df9['y_sub2']=np.abs(df9['y_sub'])
#2
df9['y_test_div']=df9['y_test']
df9.loc[df9['y_test'] == 0, 'y_test_div'] =0.00001
df9['y_subk']=(df9['y_sub2']/df9['y_test_div'])*100
dfk=df9[df9['y_subk']<ky0]
dsum=len(dfk['y_pred'])
dacc=dsum/ny_test*100

```

下面具体分析图 16-1 中的各个数据列。

- pwin、pdraw0、plost0 分别是足彩胜、平、负开盘的赔率数据，这些字段源自 x_test 测试和训练数据集，不同的数据集是完全不同的。
- y_test 是真实比赛结果，使用数字 3、1、0 分别代表胜、平、负，其中胜的结果数值 3 已经修改为 2；如果是预测，比赛还没有开始，则数值为-1。
- y_pred 是机器学习算法的预测数据，如果 fgInt=True 即整数结果模式，则这个数值为四舍五入后的整数结果，本案例和爱丽丝这些分类性质的数据分析都是整数结果模式；CCPP 数据集案例和具体股价分析属于预测具体数值，需要采用非整数结果模式。
- y_preds_r 是机器学习算法的原始预测数据，即 y_pred 四舍五入前的数值，如果是组合算法等多个机器学习算法，则该数值为当前算法的数值。
- y_sub 是原始结果和预测结果之间的差值。
- y_sub2 是 y_sub 的绝对数。
- y_test_div 是 y_test 真实比赛结果的复制值，如果原值为零，则改为 0.00001，避免出现除法分母为零的错误。
- y_subk 是 y_sub2 真实数值和预测数值的绝对差除以 y_test_div 修正后的真实数值，再除以百分比，简单来说，就是误差与原值的百分比，即误差值。误差值小于 yk0，就认为是正确值，默认误差值在 5%以内。

在图 16-1 中进行人工分析，只有第 19 行的数据列 y_subk 误差值小于 5%，其他的误差值均是 50%。再看看第 19 行的真实比赛结果，数值 y_test 等于 2，预测比赛结果数值 y_pred 也等于 2，两个字段的数值完全相同，说明第 19 行该场比赛的预测结果是正确的。

读者可以用 Excel 浏览 df9_pred.csv 文件，进行类似的分析。

通过本节的说明，对于神秘的 df9，读者应该非常清楚了。

此外还需要注意的是，在 `mx_fun8mx` 机器学习接口函数和 `zai.ai_acc_xed` 结果验证函数中，使用的部分数据字段名称类似 Python 语言中的保留字、关键词，在原始数据集中，如果字段名称相同，调用函数前，需改为其他名称，避免出现意外的错误。

16.2 盈利率分析

在金融量化回溯分析中，笔者一直强调：

对于任何机器学习算法模型，唯一有效的指标，只有盈利率。

在前面的回溯案例中，使用的也是盈利率指标，不过对于机器学习案例部分，因为程序架构问题，对 `zai.ai_acc_xed` 结果验证函数输入了多种分析数据，却没有盈利率指标。

16.2.1 案例 16-2：盈利率计算

案例 16-2 的文件名是 `zd302_df02.py`，通过对结果数据文件 `df9_pred.csv` 进行分析，计算盈利率。

案例 16-2 的程序虽然较长，但很简单，没有使用新的函数，为了说明方便，笔者对程序代码进行了分组，下面逐一介绍。

第 1 组代码：

```
#1
fss='dat/df9_pred.csv'
df=pd.read_csv(fss,index_col=False)
df9=df[df.y_test!=-1]
dn90,xn90=len(df.index),len(df9.index)
print('\n#1,df9,xn90',xn90,fss,dn90)
print(df9.head())
```

运行流程如下：

- 读取 `df9` 的结果数据文件 `dat/df9_pred.csv`，保存到变量 `df`；
- 对 `df` 变量保存的数据进行处理，过滤 `y_test` 比赛结果数据是 -1 的结果数据，新结果保存到变量 `df9`；

- 计算 df、df9 的数据长度，Pandas 一般用 len(df.index) 速度最快。

对应的输出信息是：

```
#1,df9,xn90 34706 dat/df9_pred.csv 34706
      pwin0 pdraw0 plost0 y_test y_pred y_predsr ysub ysub2
y_test_div ysubk
0    1.24    4.90     8.5     2     1  1.409777     1     1     2.0    50.0
1    1.29    5.50    10.0     2     1  1.467774     1     1     2.0    50.0
2    1.25    5.15     9.0     2     1  1.430041     1     1     2.0    50.0
3    1.30    5.25    10.0     2     1  1.464406     1     1     2.0    50.0
4    1.29    5.50     9.5     2     1  1.449774     1     1     2.0    50.0
```

由以上输出信息可以看出，dn90 和 xn90 的数值都是 34796，完全一样，这说明 df9_pred.csv 结果数据文件中的数据已经经过了初步的清洗，删除了没有开始比赛的数据。

在实盘预测时，虽然需要删除历史上没有进行的比赛数据，但对未来数日尚未开始的数据，依然需要保留，只是在计算最后结果数值时需要注意。

第 2 组代码，筛选误差值小于 5% 的数据，保存到变量 dfk：

```
#2
dfk=df9[df9.ysubk<5]
kn90=len(dfk.index)
print('\n#2,dfk,n_dfk,',kn90)
print(dfk.head())
```

对应的输出信息是：

```
#2,dfk,n_dfk, 10021
      pwin0 pdraw0 plost0 y_test y_pred y_predsr ysub ysub2
y_test_div ysubk
17    1.28    6.00    11.00     2     2  1.509956     0     0     2.0     0.0
29    1.32    5.95    11.39     2     2  1.521225     0     0     2.0     0.0
31    1.38    6.50    13.00     2     2  1.582066     0     0     2.0     0.0
99    4.20    3.65     1.62     1     1  0.984640     0     0     1.0     0.0
100   4.20    3.60     1.83     1     1  0.991637     0     0     1.0     0.0
```

由输出信息可以看出 kn90=10021，即正确数据有 10021 条，此外注意输出 dfk 的数据字段，其中 y_test 和 y_pred 字段的数值都是相同的，说明预测结果与真实数据一致。

第 3 组代码，将正确数据变量 dfk 按胜、平、负分别筛选数据，并保存到变量

df3、df1 和 df0:

```
#3

df3=dfk[dfk.y_pred==2]
df1=dfk[dfk.y_pred==1]
df0=dfk[dfk.y_pred==0]
dn3,dn1,dn0=len(df3.index),len(df1.index),len(df0.index)
print('\n#3,df310')
print('dn310,',dn3,dn1,dn0)
```

对应的输出信息是:

```
#3,df310
dn310, 1537 8399 85
```

说明在正确的比赛数据中,胜局有 1537 条,平局有 8399 条,负局有 85 条。

第 4 组代码,将总的的数据变量 df9 按胜、平、负分别筛选数据,并保存到变量 xf3、xf1 和 xf0:

```
#
#4
xf3=df9[df9.y_pred==2]
xf1=df9[df9.y_pred==1]
xf0=df9[df9.y_pred==0]
xn3,xn1,xn0=len(xf3.index),len(xf1.index),len(xf0.index)
print('\n#4,xn310',xn3,xn1,xn0)
```

对应的输出信息是:

```
#4,xn310 1736 32856 85
```

说明在总的比赛数据中,胜局有 1736 条,平局有 32856 条,负局有 85 条。

第 5 组代码,从正确的胜、平、负数据变量 df3、df1 和 df0 中,根据赔率计算收入金额:

```
#5
print('\n#5,dget310')
dget3=round(df3.pwin0.sum())
dget1=round(df1.pdraw0.sum())
dget0=round(df0.plost0.sum())
print('dget310',dget3,dget1,dget0)
```

对应的输出信息是:

```
#5,dget310
```

```
dget310 1799 28486 105
```

说明在正确的比赛数据中，胜局的收入是 1799 元，平局的收入是 28486 元，负局的收入是 105 元。

第 6 组代码，计算比赛的预测准确度：

```
#6
print('\n#6,k310')
dn9,xn9=dn3+dn1+dn0,xn3+xn1+xn0
k9=100*dn9/xn9
k3,k1,k0=100*dn3/xn3,100*dn1/xn1,100*dn0/xn0
xss='k9,{0:.2f}% ,k3,{1:.2f}% ,k1,{2:.2f}% ,k0,{3:.2f}% '.format(k9,k3,k1,k0)
print(xss)
print('dn9,xn9, ',dn9,xn9)
```

对应的输出信息是：

```
#6,k310
k9,28.90%,k3,88.54%,k1,25.56%,k0,100.00%
dn9,xn9, 10021 34677
```

说明在结果数据中：

- 比赛总的结果准确度 $k9=28.9\%$;
- 胜盘的预测准确度 $k3=88.54\%$;
- 平局的预测准确度 $k1=25.56\%$;
- 负盘的预测准确度 $k0=100\%$ 。

第 7 组代码，计算最终的盈利率：

```
#7
print('\n#7,dk310')
dk9=100*(dget3+dget1+dget0)/xn9
dk3,dk1,dk0=100*dget3/xn3,100*dget1/xn1,100*dget0/xn0
xss='$dget,k9,{0:.2f}% ,k3,{1:.2f}% ,k1,{2:.2f}% ,k0,{3:.2f}% '.format(dk9,dk3,dk1,dk0)
print(xss)
```

对应的输出信息是：

```
#7,dk310
$dget,dk9,87.64%,dk3,103.63%,dk1,86.70%,dk0,123.53%
```

说明在结果数据中：

- 比赛总的盈利率 $dk9=87.64\%$ ，实际上是亏损 $100\%-87.64\%=12.36\%$;
- 胜盘的盈利率 $dk3=103.63\%$;
- 平局的盈利率 $dk1=86.7\%$;
- 负盘的盈利率 $dk0=123.53\%$ 。

总体来看，虽然总的盈利率是亏损的，但结果数据中的胜盘盈利率和负盘盈利率都是正收益，说明该机器学习算法如果只使用胜盘和负盘的预测数据，总的收益也是正收益。

以上就是最简单的收益率计算分析，需要注意的是，以上只是针对结果数据文件 `dat/df9_pred.csv` 进行的直接分析，没有考虑博彩机构的差异，也没有考虑 `gid` 比赛场次的筛选，在进行胜盘分析时，需要通过这些数据，预先对 `df9` 结果变量进行整理，才能获得最终的正确结果。

17

第 17 章

机器学习足彩实盘分析

本章通过一个实盘案例，介绍完整的机器学习足彩实盘分析流程，包括以下内容：

- 使用机器学习算法设计回溯分析策略；
- 实时更新实盘数据，采集当天的赔率数据；
- 使用回溯程序，分析当天的赔率数据并提取预测结果。

很多《零起点 Python 大数据与量化交易》的读者，和 zwQuant 开源量化软件的用户，在刚开始的时候，总是在问：“怎么导入当天的 A 股数据进行量化回溯分析？”

这句话背后的意思是：

zwQuant 量化软件的 main 主入口在哪里？

笔者在本书中，的确没有给出一个完整的实盘数据更新加回溯分析的案例，不过有经验的读者看完全书，都能很容易地找到程序的入口。

有些水平高的读者，不到一个月的时间，就会修改程序当中的 MACD 等相对复杂的策略函数，还有读者甚至用 C++ 语言把整个 zwQuant 量化程序重新编写了一遍。

虽然水平高的读者不少，但是量化分析毕竟属于新生事物，绝大部分用户都是属于量化领域的初学者，为此，笔者特意在 TopQuant.vip 极宽量化社区发布了一篇补充文档：“zwQuant 实盘操作指南”，其网址是 <http://ziwang.com/forum.php?mod=viewthread&tid=17523>。为了避免出现以上问题，笔者在最后一章增加了一个完整的实时数据更新和回溯分析案例，便于初学者学习。

17.1 回溯主入口

在 TFB 足彩量化回溯系统中，我们特意增加了一个 `tfb_main` 主入口模块，在模块最后有一个 `main` 主函数定义：

```
def main():  
    print('main...')  
    #1  
    #main_get()  
    #2  
    #main_bt()  
    #3  
    print('main,ok')
```

其中：

- 第 1 组代码，采用默认参数，调用 `main_get` 函数，实时更新当天和前一天的足彩实盘赔率数据；
- 第 2 组代码，采用默认参数，调用 `main_bt` 主回溯接口函数，对当天抓取的实时数据进行回溯分析。

以上两个函数在调用代码前，都加了“#”注解符号进行屏蔽，这是因为：

- `main_get` 函数在下载网络数据时，需要一定的时间，一般是 5~10 分钟；
- 在实盘操作时，因为网络问题，为避免丢失数据，经常需要连续运行 2~3 次 `main_get` 函数，多下载几次赔率数据；
- `main_bt` 主回溯接口函数，因为使用不同的策略，需要进行多处的参数设置，所以一般只是作为参考模板，复制到主流程中再使用。

在实盘中，一般是把 `main_get` 数据更新函数和 `main_bt` 主回溯接口函数都放在主流程，先屏蔽回溯接口函数，运行 2~3 次 `main_get` 数据更新函数：

```
main_get()  
#main_bt()
```

再屏蔽数据更新函数，运行回溯函数：

```
#main_get()  
main_bt()
```

回溯函数只需运行一次，有经验的用户在调用回溯函数时，可以不使用默认参数，`nday` 的数值设置为 20~30，可以看到最近 20~30 天的实盘结果数据，如策略的

准确率、回报率等。

17.1.1 案例 17-1：策略 sta01

案例 17-1 的文件名是 `zd401_sta01.py`，再一次使用最简单的 `sta01` 策略，回顾一下整个回溯流程。

根据 `tfb_main` 主入口模块的 `main_bt` 函数和 `main` 主函数的相关代码，编写了案例 17-1 的代码，核心代码如下：

```
#1,set.dat
timStr=''
#
# 2#,get.data
#tfb_main.main_get('',2)
#
# 3#,backtest,main_bt
main_bt(timStr,2)
```

运行后是空数据，说明最近两天的数据没有合适的比赛推荐。

案例 17-1 的其他代码以前介绍过，不再重复，请读者自己解读。

17.1.2 结果文件解读

因为默认参数调用 `main_bt` 函数，没有数据，所以设置为 30 天的回溯周期，看看相关的运行结果，案例 17-1 的第 3 组代码改为：

```
main_bt(timStr,30)
```

运行后，会输出类似以下的信息：

```
xtfb.poolTrd,足彩推荐
```

	gid	gset	mplay	mtid	gplay	gtid	qj	qs	qr	kend	kwin	kwinrq
tweek		tplay			tsell	cid	pwin9	pdraw9	plost9	kwin_sta		
0	647699	瑞典杯	索尔纳	215	盖斯	288	-1.0	-1.0	0.0	0	-1.0	
-1.0	0	2017-02-19	2017-02-19	20:55:00	1	1.13	5.90	13.50	3.0			
1	647717	瑞典杯	埃尔夫	563	瓦纳默	3913	-1.0	-1.0	0.0	0	-1.0	
-1.0	0	2017-02-19	2017-02-19	22:55:00	1	1.15	5.80	11.50	3.0			

```

2 575970 荷甲 海牙 211 费耶诺 801 -1.0 -1.0 0.0 0 -1.0
-1.0 0 2017-02-19 2017-02-19 23:40:00 1 10.50 5.35 1.18 0.0
3 634238 英足总杯 布莱克 723 曼联 1075 -1.0 -1.0 0.0 0 -1.0
-1.0 0 2017-02-19 2017-02-20 00:10:00 1 12.00 5.25 1.17 0.0
4 574038 法甲 日尔曼 647 图卢兹 1233 -1.0 -1.0 0.0 0 -1.0
-1.0 0 2017-02-19 2017-02-20 00:55:00 1 1.08 6.75 18.00 3.0

xtfb.poolRet, 回报率汇总
      xtim  kret9  kret3  kret1  kret0  knum9  knum3  knum1  knum0
ret9 ...  ret3  ret1  ret0  nwin3  nwin1  nwin0  num3  num1  num0  cid
10 2017-02-04 0.00 0.00 0.0 0.0 0.00 0.00 0.00 0.0 0.0
0.00 ... 0.00 0.0 0.00 0.0 0.0 0.0 1.0 0.0 0.0 1
11 2017-01-27 56.00 112.00 0.0 0.0 50.00 100.00 0.0
0.0 1.12 ... 1.12 0.0 0.00 1.0 0.0 0.0 1.0 0.0 1.0 1
12 2017-01-26 89.00 73.00 0.0 137.0 75.00 66.67 0.0
100.0 3.56 ... 2.19 0.0 1.37 2.0 0.0 1.0 3.0 0.0 1.0 1
13 2017-01-24 119.00 0.00 0.0 119.0 100.00 0.00 0.0
100.0 1.19 ... 0.00 0.0 1.19 0.0 0.0 1.0 0.0 0.0 1.0 1
14      sum 93.03 89.97 0.0 104.5 81.58 80.00 0.0 87.5
35.35 ... 26.99 0.0 8.36 24.0 0.0 7.0 30.0 0.0 8.0 1

```

从输出信息可以看出：

- 回溯周期是最近 30 天，只有胜盘和负盘的数据，平局的数据为零；
- 总的现金回报率是 93%，胜盘、负盘的现金回报率分别是 89.9%和 104%；
- 总的比赛预测准确度是 81.58%，胜盘、负盘比赛预测准确度分别是 80%和 87%；
- 预测为胜盘的比赛有 30 场，其中正确的有 24 场；预测为负盘的比赛有 8 场，其中正确的有 7 场。

程序会在 log 目录下生成两个数据文件，文件名上的日期参数、内容和程序具体运行时间有关系。

- poolRet_2017-02-22.csv 是回报率分析数据文件，文件最后一条信息的日期栏是 sum，表示数据汇总信息，在分析该文件时，需要去掉该行信息。
- poolTrd_2017-02-22.csv 是推荐比赛数据记录文件，输出信息中的 poolTrd 足彩推荐就是其中的文件头数据。

以上两个文件中的数据都是按比赛时间逆序排列的，越新的比赛，排名越靠前。

17.1.3 数据字段分析

为了保持统一的学习体验，在 dat 目录下有 poolRet_2017-02-22.csv 和 poolTrd_2017-02-22.csv 两个文件的复制文件。如图 17-1 所示是 poolRet_2017-02-22.csv 回报率分析数据文件的截图。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	xtim	kret9	kret3	kret1	kret0	knum9	knum3	knum1	knum0	ret9	num9	nwin9	ret3	ret1	ret0	nwin3	nwin1	nwin0	num3	num1	num0	cid
2	2017/2/19	55.5	55.5	0	0	50	50	0	0	1.11	2	1	1.11	0	0	1	0	0	2	0	0	1
3	2017/2/18	52.5	52.5	0	0	50	50	0	0	1.05	2	1	1.05	0	0	1	0	0	2	0	0	1
4	2017/2/17	117.25	117.25	0	0	100	100	0	0	4.99	4	4	4.99	0	0	4	0	0	4	0	0	1
5	2017/2/13	118	0	0	118	100	0	0	100	1.18	1	1	0	0	1.18	0	0	1	0	0	1	1
6	2017/2/12	87.67	84.88	0	110	77.78	75	0	100	7.89	9	7	6.79	0	1.1	6	0	1	8	0	1	1
7	2017/2/11	114.83	112.75	0	119	100	100	0	100	6.89	6	6	4.51	0	2.38	4	0	2	4	0	2	1
8	2017/2/10	113.5	113.5	0	0	100	100	0	0	2.27	2	2	2.27	0	0	2	0	0	2	0	0	1
9	2017/2/9	114	0	0	114	100	0	0	100	1.14	1	1	0	0	1.14	0	0	1	0	0	1	1
10	2017/2/8	109	109	0	0	100	100	0	0	2.18	2	2	2.18	0	0	2	0	0	2	0	0	1
11	2017/2/6	108	108	0	0	100	100	0	0	1.08	1	1	1.08	0	0	1	0	0	1	0	0	1
12	2017/2/4	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1
13	2017/1/27	56	112	0	0	50	100	0	0	1.12	2	1	1.12	0	0	1	0	0	1	0	1	1
14	2017/1/26	89	73	0	137	75	66.67	0	100	3.56	4	3	2.19	0	1.37	2	0	1	3	0	1	1
15	2017/1/24	119	0	0	119	100	0	0	100	1.19	1	1	0	0	1.19	0	0	1	0	0	1	1
16	sum	93.03	89.97	0	104.5	81.58	80	0	87.5	35.35	38	31	26.99	0	8.36	24	0	7	30	0	8	1

图 17-1 回报率分析数据文件截图

图 17-1 中的相关字段介绍如下。

- xtim，比赛日期。
- kret9，总的现金回报率。
- kret3、kret1、kret0，胜盘、平局、负盘的现金回报率。
- knum9，总的比赛预测准确度。
- knum3、knum1、knum0，胜盘、平局、负盘的比赛预测准确度。
- ret9，总的现金收入金额。
- num9，总的比赛场次。
- nwin9，正确预测的比赛场次。
- ret3、ret1、ret0，胜盘、平局、负盘的现金收入金额。
- nwin3、nwin1、nwin0，胜盘、平局、负盘正确预测的比赛场次。
- num3、num1、num0，胜盘、平局、负盘预测的比赛场次。
- cid，结算使用的博彩机构代码，一般是 1，代表中国体彩中心。

如图 17-2 所示是推荐比赛数据记录文件的截图。

图 17-2 中的相关字段介绍如下。

- gid，比赛场次 id 编码。
- gset，联赛名称。
- mplay，主队名称。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	gid	gsset	aplay	atid	gplay	gtid	qj	qs	qr	kend	kwin	kwinr	tweek	tplay	tsell	cid	pwin9	pdraw9	plost9	kwin_sta	
2	647099	瑞典杯	奈尔纳	215	墨斯	288	-1	-1	0	0	-1	-1	0	2017/2/19	2017/2/19 20:55	1	1.13	5.9	13.5	3	
3	647177	瑞典杯	埃尔夫	565	瓦纳默	5913	-1	-1	0	0	-1	-1	0	2017/2/19	2017/2/19 22:55	1	1.15	5.8	11.5	3	
4	575970	德甲	海牙	211	费耶诺	801	-1	-1	0	0	-1	-1	0	2017/2/19	2017/2/19 23:40	1	10.5	5.35	1.15	0	
5	634238	英足总杯	布莱克	723	曼联	1075	-1	-1	0	0	-1	-1	0	2017/2/19	2017/2/20 0:10	1	12	5.25	1.17	0	
6	574038	德甲	日尔曼	647	图卢兹	1233	-1	-1	0	0	-1	-1	0	2017/2/19	2017/2/20 0:55	1	1.08	6.75	18	3	
7	630300	巴圣杯	林斯	6460	帕尔梅	1567	-1	-1	0	0	-1	-1	0	2017/2/19	2017/2/20 0:55	1	8.55	4.7	1.25	0	
8	610616	意甲	亚特兰	1308	克罗托	243	1	0	0	1	3	-1	6	2017/2/19	2017/2/19 0:55	1	1.11	6.1	15.5	3	
9	630298	巴圣杯	奥托斯	472	费罗维	8626	0	1	0	1	0	-1	6	2017/2/19	2017/2/19 0:55	1	1.17	5.32	11.5	3	
10	634232	英足总杯	伯恩利	700	林肯城	1372	0	1	0	1	0	-1	6	2017/2/18	2017/2/18 20:25	1	1.15	5.5	13	3	
11	607063	西甲	皇马	883	西牙人	1283	2	0	0	1	3	-1	6	2017/2/18	2017/2/18 23:10	1	1.05	8.15	18.5	3	
12	628583	欧罗巴	毕尔巴	690	希姆人	2510	3	2	0	1	3	-1	4	2017/2/17	2017/2/16 23:55	1	1.15	5.5	12.75	3	
13	628590	欧罗巴	曼联	1075	埃斯安	494	3	0	0	1	3	-1	4	2017/2/17	2017/2/16 23:55	1	1.1	6.35	10	3	
14	630294	巴圣杯	帕尔梅	1567	圣贝尔	0461	2	0	0	1	3	-1	4	2017/2/17	2017/2/16 23:55	1	1.15	5.4	13.5	3	
15	634471	巴超杯	弗拉门	1577	米美洲	6212	1	0	0	1	3	-1	4	2017/2/17	2017/2/16 23:55	1	1.29	4.4	8	3	
16	610610	意甲	卡利亚	917	尤文	1330	0	2	0	1	0	-1	0	2017/2/13	2017/2/13 0:55	1	10.5	5.35	1.15	0	
17	575958	德甲	门兴	540	霍夫丹	518	2	0	0	1	3	-1	0	2017/2/12	2017/2/12 21:25	1	1.07	7.25	17.5	3	
18	610608	意甲	国米	846	恩波利	788	2	0	0	1	3	-1	0	2017/2/12	2017/2/12 21:55	1	1.16	5.42	12	3	
19	634265	苏足总杯	流浪者	827	摩顿	3542	2	1	0	1	3	-1	0	2017/2/12	2017/2/12 22:55	1	1.15	5.91	11	3	
20	575960	德甲	埃因德	378	德勒	1275	3	0	0	1	3	-1	0	2017/2/12	2017/2/12 23:40	1	1.19	5.35	9.75	3	
21	575956	德甲	费耶诺	801	格罗宁	824	2	0	0	1	3	-1	6	2017/2/12	2017/2/12 0:55	1	1.13	6	13	3	
22	574021	德甲	摩纳哥	1105	梅斯	1084	5	0	0	1	3	-1	6	2017/2/12	2017/2/12 0:55	1	1.09	6.8	15.5	3	
23	573309	比甲	根特	839	欧本	2100	0	1	0	1	0	-1	6	2017/2/12	2017/2/12 0:55	1	1.21	4.95	10	3	
24	607055	西甲	奥萨苏	622	塞马	883	1	3	0	1	0	-1	6	2017/2/12	2017/2/12 0:55	1	13.75	6.8	1.1	0	
25	630285	巴圣杯	科林蒂	1572	圣安德	2204	0	2	0	1	0	-1	6	2017/2/12	2017/2/12 0:55	1	1.2	4.8	11.5	3	
26	572964	英超	阿森纳	554	赫尔城	872	2	0	0	1	3	-1	6	2017/2/11	2017/2/11 20:25	1	1.15	5.77	11.5	3	
27	606340	德甲	柏林赫	2013	柏林	664	0	0	0	1	0	-1	6	2017/2/11	2017/2/11 22:05	1	6.1	4.7	1.24	0	

图 17-2 推荐比赛数据记录文件截图

- mtid, 主队代码 id。
- gplay, 客队名称。
- gtid, 客队代码 id。
- qj, 进球数目。
- qs, 失球数目。
- qr, 让球数目。
- kend, 比赛结束标志, -1 为未结束。
- kwin, 比赛结果, 以 3、1、0 表示胜、平、负。
- kwinrq, 让球比赛结果, 以 3、1、0 表示胜、平、负。
- tweek, 比赛日期为周日。
- tplay, 比赛日期。
- tsell, 彩票销售截止时间。
- cid, 结算使用的博彩机构代码, 一般是 1, 代表中国体彩中心。
- pwin9、pdraw9、plost9, 代码 cid 的收盘胜、平、负的赔率数据。
- kwin_sta, 程序预测的比赛结果数据, 以 3、1、0 表示胜、平、负。

17.2 机器学习与回溯分析

案例 17-1 简单回顾了足彩量化回溯分析的整个运行流程, 下面结合机器学习算法, 对实盘足彩数据进行具体的分析, 并推荐预测的比赛结果。

在开始使用机器学习算法之前, 看看 15 章中各种机器学习算法模型的基本测试。

如图 17-3 所示是机器学习算法测试数据对比图，汇总了案例 15-5 到案例 15-9 的测试数据。

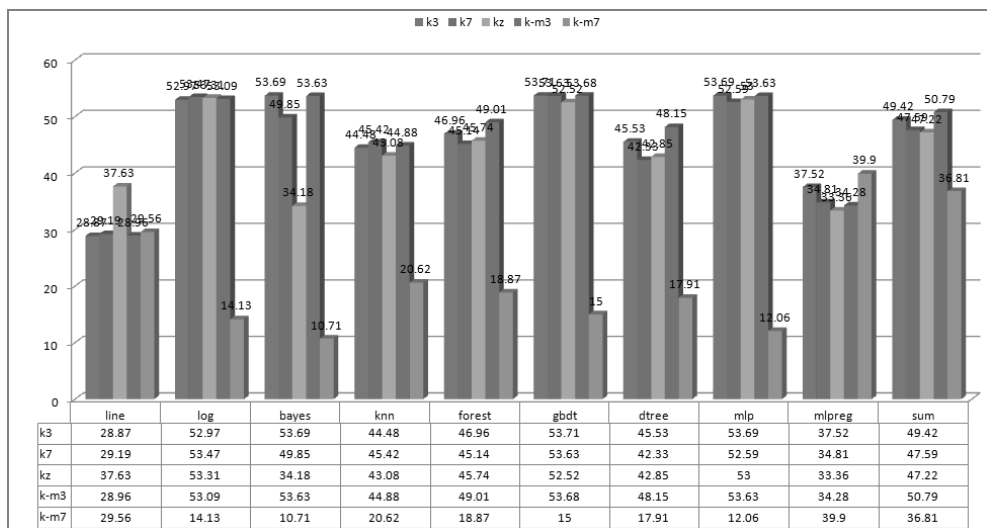


图 17-3 机器学习算法测试数据对比图

由图 17-3 可以看出：

- k3、k7、kz 分别是机器学习算法单一年度赔率数据 3 字段、7 字段、全字段模型的测试数据；
- k-m3、k-m7 分别是机器学习算法多年累计赔率数据的 3 字段、7 字段模型的测试数据；
- 7 字段模型测试数据与其他模型测试数据差别较大；
- 删除 7 字段模型测试数据 k-m7，在其他测试模型中，Log 逻辑回归算法、GBDT 迭代决策树算法、MLP 神经网络算法的测试数据准确度都在 50% 以上；
- Line 线性回归算法虽然整体测试准确度偏低，但其 7 字段模型测试数据的准确度居于第二；
- 各个字段的组合算法总的准确度，均低于组合当中的最高单一算法的准确度。

综上所述，在进行足彩实盘分析时，使用准确度相对较高、运行速度也较快的 Log 回归算法，训练模型采用年度模型 p7year2016 的 7 字段系列机器学习算法模型。

17.2.1 案例 17-2: Log 回归策略足彩分析

案例 17-2 的文件名是 `zd402_log01.py`，它是基于 Log 回归机器学习算法的足彩赔率实盘分析案例，下面对代码分组进行讲解。

```
#
#1,set.dat
timStr=''

#
# 1#,get.data
#main_get('',2)

#
# 2#,backtest,main_bt
main_ai_bt(timStr,2)
```

函数代码很简单，与案例 17-1 类似，只是调用的回溯接口函数由 `main_bt` 变成了机器学习算法版本的 `main_ai_bt`，这是因为机器学习算法部分的参数有些特殊。

机器学习算法版本的 `main_ai_bt` 回溯接口函数，在 `tfb_main` 主入口模块中也有，程序代码与 `main_bt` 函数类似，只是在整数调用 `bt_main` 回溯测试主函数前，多了一组机器学习算法的参数设置代码：

```
#3.a-----ai.init
    zai.xmodel={}
    xtfb.ai_mxFFN0=rs0+'mllib/p7y2016_'
    xtfb.ai_mx_sgn_lst=['log']

    xtfb.ai_ysgn='kwin'
    xtfb.ai_xlst=['cid','pwin0','pdraw0','plost0','pwin9','pdraw9',
'plost9']
    #
    zai.ai_f_mxRdlst(xtfb.ai_mxFFN0,xtfb.ai_mx_sgn_lst)

#3.b
    xtfb.kcid='1' #cn,3=bet365
    tfbt.bt_main(xtfb,timStr)
```

3.a 组代码很简单，在 `xtfb` 变量中，使用的内部代码都是约定模式，读者对照前

面的机器学习算法的案例，很容易就可以看懂。

在案例 17-2 中，使用的策略是 log 回归策略：

```
xtfb.funPre=tfsty.sta00_pre #bt_1dayMain
xtfb.funSta=tfsty.sta_ai_log01
xtfb.preVars=[]
xtfb.staVars=[99,99,99]
```

Log 回归策略无需进行数据预处理，所以对应的数据预处理函数是 tfsty.sta00_pre 空函数。

使用的策略函数是 tfsty.sta_ai_log01，其对应的参数是：

```
xtfb.staVars=[99,99,99]
```

在 Log 回归策略函数代码中，这些参数表示胜、平、负的匹配程度，按百分比格式，参数设置为 99。在策略结果数据中，胜、平、负的最终数据，都要大于 99% 或者 100% 正确时才会采用。

17.2.2 Log 回归策略函数

Log 回归策略函数位于 tfb_strategy 极宽策略模块库，函数代码是：

```
def sta_ai_log01(xtfb,df):
    #1
    xkwin,k00,k10,k30=-9,xtfb.staVars[0] ,xtfb.staVars[1],xtfb.
staVars[2]
    ysgn='kwin' #xtfb.ai_ysgn
    #2
    df[ysgn]=df[ysgn].astype(str)
    df[ysgn].replace('3','2', inplace=True)
    #3
    df[ysgn]=df[ysgn].astype(int)
    #4
    xtfb.ai_xdat,xtfb.ai_ydat= df[xtfb.ai_xlst],df[ysgn]
    #5
    msgn=xtfb.ai_mx_sgn_lst[0] #'log'
    mx=zai.xmodel[msgn]
    dacc,df9=zai.mx_fun8mx(mx, xtfb.ai_xdat,xtfb.ai_ydat,yk0=1,
fgInt=True) #,fgDebug=True
```

```

        #print('\n log01,dacc,',dacc)
        #6
        df3,df1,df0=df9[df9['y_pred']==2],df9[df9['y_pred']==1],
df9[df9['y_pred']==0]
        dn3,dn1,dn0=len(df3.index),len(df1.index),len(df0.index)
        #7
        dn9,dsum=max(dn3,dn1,dn0),sum([dn3,dn1,dn0])
        #8
        if dsum>0:
            dk3,dk1,dk0=dn3/dsum*100,dn1/dsum*100,dn0/dsum*100
            if (dn3==dn9)and(dk3>k30):xkwin=3
            elif (dn1==dn9)and(dk1>k10):xkwin=1
            elif (dn0==dn9)and(dk0>k00):xkwin=0
            #
            #yk310=df9['y_test'][0]
            #xs0='@log01,{0}#{1},{xk,gid,{2},dsum.{3},dn310,{4},{5},{6},
dk310,{7:.1f}%,{8:.1f}%,{9:.1f}%'
            #xss=xss0.format(xkwin,yk310,xtfb.kgid,dsum,dn3,dn1,dn0,dk3,
dk1,dk0)

            #print(xss)

        #9
        return xkwin

```

Log 回归策略函数的代码看起来很长，其实很简单，已经进行了分组，下面按运行流程分组讲解。

策略函数定义中的 **df** 变量是 **gid** 编码代表的比赛场次中所有机构的赔率数据，如图 17-4 所示为欧赔数据网页截图。

第 1 组代码，设置 **xkwin** 参数和相关变量：

```

#1
xkwin,k00,k10,k30=-9,xtfb.staVars[0],xtfb.staVars[1],xtfb.staVars[2]
ysgn='kwin' #xtfb.ai_ysgn

```

第 2 组代码，修改 **df** 的 **ysgn** 数据列即 **kwin** 比赛结果代码，将胜盘代码 3 改为 2，这是机器学习算法约定的，这里的数据改变不影响最终的回报率结果计算：

```

#2
df[ysgn]=df[ysgn].astype(str)
df[ysgn].replace('3','2', inplace=True)

```

数据分析 投注分析 百家欧赔 让球指数 亚盘对比 大小指数 比分指数 走势分析 技术统计											
筛选	全部公司	即时欧赔			即时胜率			返还率	即时赢利		
序号	赔率公司	胜	平	负	胜	平	负	值	胜	平	负
1	竞彩官方	1.68	3.65	3.85	52.72%	24.27%	23.01%	88.58%	0.88	0.92	0.87
		1.62	3.70	4.15	54.70%	23.95%	21.35%	88.61%	0.85	0.93	0.94
2	威廉希尔	1.85	3.75	4.00	51.13%	25.22%	23.65%	94.59%	0.97	0.95	0.90
		1.85	3.75	4.00	51.13%	25.22%	23.65%	94.59%	0.97	0.95	0.90
3	澳门	1.82	3.60	3.45	49.19%	24.87%	25.95%	89.52%	0.95	0.91	0.78
		1.75	3.60	3.75	51.21%	24.89%	23.90%	89.62%	0.91	0.91	0.85
4	立博	2.10	3.60	3.20	44.65%	26.05%	29.30%	93.77%	1.10	0.91	0.72
		1.83	3.60	4.00	50.87%	25.86%	23.27%	93.09%	0.96	0.91	0.90
5	Bet365	2.00	3.60	3.50	47.01%	26.12%	26.87%	94.03%	1.04	0.91	0.79
		1.83	3.60	4.50	52.22%	26.54%	21.24%	95.56%	0.96	0.91	1.01
6	Interwetten	1.75	3.65	3.75	51.38%	24.64%	23.98%	89.92%	0.91	0.92	0.85
		1.75	3.60	3.90	51.68%	25.12%	23.19%	90.45%	0.91	0.91	0.88
7	SNAI	1.95	3.50	3.45	47.12%	26.25%	26.63%	91.88%	1.02	0.88	0.78
		1.75	3.65	4.00	52.17%	25.01%	22.82%	91.29%	0.91	0.92	0.90
8	皇冠	1.91	3.70	3.25	47.53%	24.54%	27.93%	90.78%	1.00	0.93	0.73
		1.81	3.70	4.35	52.49%	25.68%	21.84%	95.00%	0.94	0.93	0.98
9	易胜博	1.83	3.50	3.70	49.57%	25.92%	24.52%	90.71%	0.96	0.88	0.83
		1.82	3.70	3.60	50.06%	24.63%	25.31%	91.12%	0.95	0.93	0.81
10	伟德	2.00	3.60	3.70	47.71%	26.50%	25.79%	95.42%	1.04	0.91	0.83
		1.80	3.75	4.40	52.94%	25.41%	21.66%	95.28%	0.94	0.95	0.99
11	Bwin	1.80	3.70	3.90	51.33%	24.97%	23.69%	92.40%	0.94	0.93	0.88
		1.80	3.70	3.90	51.33%	24.97%	23.69%	92.40%	0.94	0.93	0.88
12	Gamebookers	1.80	3.70	3.90	51.33%	24.97%	23.69%	92.40%	0.94	0.93	0.88
		1.80	3.70	3.90	51.33%	24.97%	23.69%	92.40%	0.94	0.93	0.88

图 17-4 欧赔数据网页截图

第 3 组代码，把 df 的 ysgn 数据列格式改为整数：

```
#3
df[ysgn]=df[ysgn].astype(int)
```

第 4 组代码，读取测试数据集：

```
#4
xtfb.ai_xdat,xtfb.ai_ydat= df[xtfb.ai_xlst],df[ysgn]
```

第 5 组代码，调用机器学习算法，生成预测数据，结果保存在 df9 中：

```
#5
msgn=xtfb.ai_mx_sgn_lst[0] #'log'
mx=zai.xmodel[msgn]
dacc,df9=zai.mx_fun8mx(mx, xtfb.ai_xdat,xtfb.ai_ydat,yk0=1,fgInt=True)
#,fgDebug=True
#print('\n log01,dacc,',dacc)
```

第 6 组代码，提取结果数据变量 df9 中胜盘、平局、负盘的结果数据：

```
#6
```

```
df3,df1,df0=df9[df9['y_pred']==2],df9[df9['y_pred']==1],df9[df9['y_pred']==0]
```

```
dn3,dn1,dn0=len(df3.index),len(df1.index),len(df0.index)
```

第 7 组代码，计算结果中最大的数值 **dn9** 和结果总数 **dsum**：

```
#7
```

```
dn9,dsum=max(dn3,dn1,dn0),sum([dn3,dn1,dn0])
```

第 8 组代码，如果结果总数 **dsum** 大于零，则计算胜盘、平局、负盘的比率，根据策略参数计算最后的预测结果 **xkwin**：

```
#8
```

```
if dsum>0:
```

```
    dk3,dk1,dk0=dn3/dsum*100,dn1/dsum*100,dn0/dsum*100
```

```
    if (dn3==dn9)and(dk3>k30):xkwin=3
```

```
    elif (dn1==dn9)and(dk1>k10):xkwin=1
```

```
    elif (dn0==dn9)and(dk0>k00):xkwin=0
```

```
    #
```

```
    #yk310=df9['y_test'][0]
```

17.2.3 案例 17-3：30 天 Log 回归策略足彩分析

案例 17-2 是基于 2 天的回溯周期分析，回溯结果没有匹配的数据，为了避免混乱，我们特意增加了一个 30 天的 Log 回归机器学习算方案例。

案例 17-3 的文件名是 `zd403_log02.py`，是 30 天基于 Log 回归机器学习算法的足彩赔率实盘分析案例，核心代码如下：

```
#
```

```
#1,set.dat
```

```
timStr=''
```

```
#
```

```
# 1#,get.data
```

```
#main_get('',2)
```

```
#
```

```
# 2#,backtest,main_bt
```

```
main_ai_bt(timStr,30)
```


函数代码很简单，与案例 17-2 类似，只是调用的参数改为 30，即采用的是 30 天回溯分析模式，这个也是熟练用户的常用模式，可以知道最近一个月的实盘回报情况。

案例 17-3 的部分运行结果如下：

```
#4,result.anz

xtfb.poolTrd,足彩推荐
      gid gset mplay mtid gplay gtid  qj  qs  qr kend kwin kwinrq
tweek      tplay      tsell cid pwin9 pdraw9 plost9 kwin_sta
  0 580051  荷乙  埃青年  5468  布雷达  350 -1.0 -1.0 0.0  0 -1.0
-1.0  1 2017-02-20 2017-02-20 23:55:00  1  2.45  3.30  2.39  3.0
  1 596660  德乙  慕尼黑  1121  纽伦堡  1138 -1.0 -1.0 0.0  0 -1.0
-1.0  1 2017-02-20 2017-02-20 23:55:00  1  2.30  3.15  2.65  3.0
  2 607062  西甲  马拉加  1056  拉斯帕  965 -1.0 -1.0 0.0  0 -1.0
-1.0  1 2017-02-20 2017-02-20 23:55:00  1  1.82  3.25  3.70  3.0
  3 574614  法乙  奥尔良  3958  朗斯  979 -1.0 -1.0 0.0  0 -1.0
-1.0  1 2017-02-20 2017-02-20 23:55:00  1  2.92  2.61  2.48  0.0
  4 577920  英冠  纽卡  1137  维拉  555 -1.0 -1.0 0.0  0 -1.0
-1.0  1 2017-02-20 2017-02-20 23:55:00  1  1.47  3.65  5.70  3.0

xtfb.poolRet,回报率汇总
      xtim  kret9 kret3 kret1 kret0 knum9 knum3 knum1 knum0
ret9 ...  ret3 ret1 ret0 nwin3 nwin1 nwin0 num3 num1 num0 cid
  17 2017-01-27  14.00 56.00  0.0  0.00 12.50 50.00  0.0  0.00
1.12 ...  1.12  0.0  0.00  1.0  0.0  0.0  2.0  0.0  6.0  1
  18 2017-01-26  87.92 91.80  0.0  82.10 56.00 60.00  0.0  50.00
21.98 ...  13.77  0.0  8.21  9.0  0.0  5.0  15.0  0.0  10.0  1
  19 2017-01-25  51.24 38.93  0.0 108.67 35.29 28.57  0.0  66.67
8.71 ...  5.45  0.0  3.26  4.0  0.0  2.0  14.0  0.0  3.0  1
  20 2017-01-24 118.67 97.00  0.0 162.00 66.67 50.00  0.0
100.00 10.68 ...  5.82  0.0  4.86  3.0  0.0  3.0  6.0  0.0
3.0  1
  21      sum  90.81 93.24  0.0  83.00 54.89 55.86  0.0  51.75
436.82 ...  342.20  0.0  94.62 205.0  0.0  59.0 367.0  0.0 114.0  1
```

程序会在 log 目录下生成两个数据文件，文件名上的日期参数、内容和程序具体运行时间有关系。

- poolRet_2017-02-22.csv 是回报率分析数据文件，文件最后一条日期栏是 sum，表示该条信息是数据汇总，分析该文件时，需要去掉该行信息。
- poolTrd_2017-02-22.csv 是推荐比赛数据记录文件，输出信息中的 poolTrd 足彩推荐就是其中的文件头数据。

17.2.4 数据文件分析

为了保持统一的学习体验，把 poolRet_2017-02-22.csv 和 poolTrd_2017-02-22.csv 两个数据文件复制到 dat 目录下，并分别改名。

- log_poolRet_2017-02-22.csv 为 Log 逻辑回归算法的回报率分析数据文件，如图 17-5 所示。
- log_poolTrd_2017-02-22.csv 为 Log 逻辑回归算的推荐比赛数据记录文件，如图 17-6 所示。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	xtia	kret9	kret3	kret1kret0	knum9	knum3	knum1	knum0	ret9	num9	num3	ret3	ret1	ret0	num9	num3	ret3	ret1	ret0	num9	num3	ret3
2	2017/2/19	97.91	90.97	0	148.25	60.61	55.17	0	100	32.31	33	20	26.4	0	5.93	16	0	4	29	0	4	1
3	2017/2/18	90.33	93.24	0	85	52.63	53.66	0	50	51.83	57	30	38.2	0	13.6	22	0	8	41	0	16	1
4	2017/2/17	90.38	92.08	0	85.25	56.25	58.33	0	50	14.46	16	9	11.1	0	3.41	7	0	2	12	0	4	1
5	2017/2/16	90.5	100.56	0	0	60	66.67	0	0	9.05	10	6	9.05	0	0	6	0	0	9	0	1	1
6	2017/2/15	82.87	97.46	0	24.5	50	58.33	0	16.67	24.86	30	15	23.4	0	1.47	14	0	1	24	0	6	1
7	2017/2/14	135.43	129	0	151.5	85.71	80	0	100	9.48	7	6	6.45	0	3.03	4	0	2	5	0	2	1
8	2017/2/13	108.45	136.23	0	68.33	63.64	76.92	0	44.44	23.86	22	14	17.7	0	6.15	10	0	4	13	0	9	1
9	2017/2/12	99.23	97.34	0	105.67	62.26	60.98	0	66.67	52.59	53	33	39.9	0	12.68	25	0	8	41	0	12	1
10	2017/2/11	89.37	87.55	0	100.67	52.31	50	0	66.67	58.09	65	34	49	0	9.06	28	0	6	56	0	9	1
11	2017/2/10	73.6	73.6	0	0	60	60	0	0	3.68	5	3	3.68	0	0	3	0	0	5	0	0	1
12	2017/2/9	110.06	101.6	0	152.33	66.67	60	0	100	19.81	18	12	15.2	0	4.57	9	0	3	15	0	3	1
13	2017/2/8	87.31	107.41	0	49.33	53.85	64.71	0	33.33	22.7	26	14	18.3	0	4.44	11	0	3	17	0	9	1
14	2017/2/7	79.2	101.5	0	34.6	46.67	60	0	20	11.88	15	7	10.2	0	1.73	6	0	1	10	0	5	1
15	2017/2/6	88	110	0	0	60	75	0	0	8.8	10	6	8.8	0	0	6	0	0	8	0	2	1
16	2017/2/5	81.95	79.44	0	92	45	43.75	0	50	16.39	20	9	12.7	0	3.68	7	0	2	16	0	4	1
17	2017/2/4	91.08	72.27	0	194.5	46.15	36.36	0	100	11.84	13	6	7.95	0	3.89	4	0	2	11	0	2	1
18	2017/1/28	103.18	100.28	0	116.25	59.09	55.56	0	75	22.7	22	13	18.1	0	4.65	10	0	3	18	0	4	1
19	2017/1/27	14	56	0	0	12.5	50	0	0	1.12	8	1	1.12	0	0	1	0	0	2	0	6	1
20	2017/1/26	87.92	91.8	0	82.1	56	60	0	50	21.98	25	14	13.8	0	8.21	9	0	5	15	0	10	1
21	2017/1/25	51.24	38.93	0	108.67	35.29	28.57	0	66.67	8.71	17	6	5.45	0	3.26	4	0	2	14	0	3	1
22	2017/1/24	118.67	97	0	182	66.67	50	0	100	10.68	9	6	5.82	0	4.80	3	0	3	6	0	3	1
23	sum	90.81	93.24	0	83	54.89	55.86	0	51.75	436.82	481	264	342	0	94.62	205	0	59	367	0	114	1

图 17-5 回报率分析数据文件截图

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	gid	gset	aplay	atid	gplay	gtid	qj	qs	qr	kend	kwinkwin	qrqweek	tplay	tsell	cid	pwin9	pdrow9	plost9	kwin_sta			
2	580051	荷乙	埃青年	5468	布露达	350	-1	-1	0	-1	-1	1	2017/2/20	#####	1	2.45	3.3	2.39	3			
3	596060	荷乙	葛伦堡	1121	伯明翰	1138	-1	-1	0	-1	-1	1	2017/2/20	#####	1	2.3	3.15	2.65	3			
4	607062	荷甲	马加比	1056	拉斯帕	965	-1	-1	0	-1	-1	1	2017/2/20	#####	1	1.82	3.25	3.7	3			
5	574614	法乙	奥尔良	3958	朗斯	979	-1	-1	0	-1	-1	1	2017/2/20	#####	1	2.92	2.61	2.48	0			
6	577920	英冠	绍卡	1137	维拉	555	-1	-1	0	-1	-1	1	2017/2/20	#####	1	1.47	3.65	5.7	3			
7	609792	葡超	马里迪	1050	葡国民	1099	-1	-1	0	-1	-1	1	2017/2/20	#####	1	1.72	3.15	4.35	3			
8	581461	澳超	中央海岸	3616	墨尔本	3615	0	3	1	0	-1	0	2017/2/19	#####	1	5.8	4.3	1.38	0			
9	607064	荷甲	社会	885	比利时	683	-1	-1	0	-1	-1	0	2017/2/19	#####	1	1.75	3.2	4.1	3			
10	610619	荷甲	博尔尼	707	国米	846	-1	-1	0	-1	-1	0	2017/2/19	#####	1	5.15	3.5	1.54	0			
11	575963	荷甲	威廉	1248	阿贾克斯	224	-1	-1	0	-1	-1	0	2017/2/19	#####	1	4.9	3.25	2.05	0			
12	596052	荷乙	比勒费	682	圣保利	495	-1	-1	0	-1	-1	0	2017/2/19	#####	1	2.25	2.85	3	3			
13	596055	荷乙	德雷斯	4654	汉诺威	862	-1	-1	0	-1	-1	0	2017/2/19	#####	1	2.38	3.25	2.5	3			
14	574308	苏超	基马诺	888	阿伯丁	527	-1	-1	0	-1	-1	0	2017/2/19	#####	1	7.65	4.6	1.28	0			
15	647699	瑞典杯	索尔纳	215	盖斯	288	-1	-1	0	-1	-1	0	2017/2/19	#####	1	1.13	5.9	13.5	3			
16	630303	巴圣锦	圣安德烈	2204	圣贝尔纳多	6461	-1	-1	0	-1	-1	0	2017/2/19	#####	1	2.22	2.95	2.94	3			
17	575969	荷甲	鹿特丹	518	格罗宁根	834	-1	-1	0	-1	-1	0	2017/2/19	#####	1	2.35	3.3	2.5	3			
18	575968	荷甲	维迪斯	1255	阿贾克斯	540	-1	-1	0	-1	-1	0	2017/2/19	#####	1	4.5	3.6	1.59	0			
19	580050	荷乙	马斯特里赫特	349	埃因霍温	1797	-1	-1	0	-1	-1	0	2017/2/19	#####	1	1.69	3.75	3.7	3			
20	573320	比甲	列日	691	根特	839	-1	-1	0	-1	-1	0	2017/2/19	#####	1	2.05	3.15	3.1	3			
21	634237	英足总杯	富勒姆	811	热刺	1238	-1	-1	0	-1	-1	0	2017/2/19	#####	1	3.8	3.7	1.68	0			

图 17-6 推荐比赛数据记录文件截图

由图 17-5 和图 17-6 中的信息可以看出：

- 回溯周期是最近 30 天，只有胜盘、负盘数据，平局数据为零；
- 总的现金回报率是 90.8%，胜盘、负盘的现金回报率分别是 93.24%和 83%；
- 总共预测了 481 场比赛，其中正确的有 264 场，总的比赛预测准确度是 54.89%；
- 总的比赛预测准确度是 54.89%，胜盘、负盘比赛预测准确度分别是 55.9%和 51.8.%；
- 预测是胜盘的比赛有 367 场，其中正确的有 205 场，准确度是 55.9%；
- 预测是负盘的比赛有 114 场，其中正确的有 59 场，准确度是 51.8%。

17.2.5 足彩推荐

注意，在比赛结果中，最新的数据特别是比赛日期是动态日期或者未来几日的，kwin 数值为-1，表示尚未开始的比赛，其中的 kwin_sta 就是程序推荐的比赛结果，将胜、平、负按数字 3、1、0 表示，kwin_sta 中的 sta 是英文单词 strategy 的缩写，是策略的意思。

轻松注册成为博文视点社区用户(www.broadview.com.cn),您即可享受以下服务。

- **提交勘误** 您对书中内容的修改意见可在【提交勘误】处提交,若被采纳,将获赠博文视点社区积分(在您购买电子书时,积分可用来抵扣相应金额)。
- **与我们交流** 在页面下方【读者评论】处留下您的疑问或观点,与我们和其他读者一同学习交流。

页面入口: <http://www.broadview.com.cn/31074>

